NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DRFM CORDIC PROCESSOR AND SEA CLUTTER MODELING FOR ENHANCING STRUCTURED FALSE TARGET SYNTHESIS

by

Pak Siang Ang

September 2017

Thesis Advisor: Phillip E. Pace
Co-Advisor: Douglas J. Fouts

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
DRFM CORDIC PROCESSOR AND SEA CLUTTER MODELING FOR ENHANCING STRUCTURED FALSE TARGET SYNTHESIS

5. FUNDING NUMBERS

6. AUTHOR(S) Pak Siang Ang

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Center for Joint Services Electronic Warfare,
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NPS, Consortium for Robotics and Unmanned Systems Education and Research (CRUSER)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

In this thesis, we investigate two critical components of a digital-image synthesizer electronic warfare architecture that can be used to infuse false targets into high-range resolution profiling radars. The first investigation encompasses the design of an in-phase and quadrature (I/Q) converter based on a CORDIC (Coordinate Rotation Digital Computer) algorithm. Mathematical modeling is used to examine the accuracy of converting a digitized radar signal I/Q sample into a corresponding five-bit binary phase angle. Results obtained from MATLAB show that 18 CORDIC iterations are required to achieve accuracy at 5.625º. The resulting design was implemented using the Verilog hardware description language. The second investigation concerns generating sea clutter to impose on the false target. The mean-power return of the sea clutter is calculated using the average power of the radar-cross section derived from the Naval Research Laboratory sea clutter model. The modulation coefficients for the sea clutter were generated using the fluctuating power returns and Doppler spectra generated using a random KA distribution. The coefficients for several sea states were generated using MATLAB. Results show that the correct sea clutter model can effectively add realism to the false target image.

14. SUBJECT TERMS inverse synthetic aperture radar, sea clutter modeling, CORDIC, Coordinate Rotation Digital Computer, Digital Image Synthesizer, DRFM, digital radio frequency memory, electronic attack	**15. NUMBER OF PAGES** 137
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DRFM CORDIC PROCESSOR AND SEA CLUTTER MODELING FOR ENHANCING STRUCTURED FALSE TARGET SYNTHESIS

Pak Siang Ang
Military Expert 5, Republic of Singapore Air Force
B.Eng., National University of Singapore, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2017

Approved by: Phillip E. Pace
Thesis Advisor

Douglas J. Fouts
Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, we investigate two critical components of a digital-image synthesizer electronic warfare architecture that can be used to infuse false targets into high-range resolution profiling radars. The first investigation encompasses the design of an in-phase and quadrature (I/Q) converter based on a CORDIC (Coordinate Rotation Digital Computer) algorithm. Mathematical modeling is used to examine the accuracy of converting a digitized radar signal I/Q sample into a corresponding five-bit binary phase angle. Results obtained from MATLAB show that 18 CORDIC iterations are required to achieve accuracy at 5.625^{o}. The resulting design was implemented using the Verilog hardware description language. The second investigation concerns generating sea clutter to impose on the false target. The mean-power return of the sea clutter is calculated using the average power of the radar-cross section derived from the Naval Research Laboratory sea clutter model. The modulation coefficients for the sea clutter were generated using the fluctuating power returns and Doppler spectra generated using a random KA distribution. The coefficients for several sea states were generated using MATLAB. Results show that the correct sea clutter model can effectively add realism to the false target image.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ADC	analog-to-digital converter
CORDIC	Coordinate Rotation Digital Computer
DIS	Digital Imaging Synthesizer
DRFM	Digital RF Memory
EM	electromagnetic
ISAR	inverse synthetic aperture radar
LOS	line-of-sight
LUT	look-up table
MSB	most significant bit
NPS	Naval Postgraduate School
NRL	Naval Research Laboratory
PRI	pulse-repetition interval
RCS	radar-cross section
RF	radio frequency
SAR	synthetic aperture radar

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

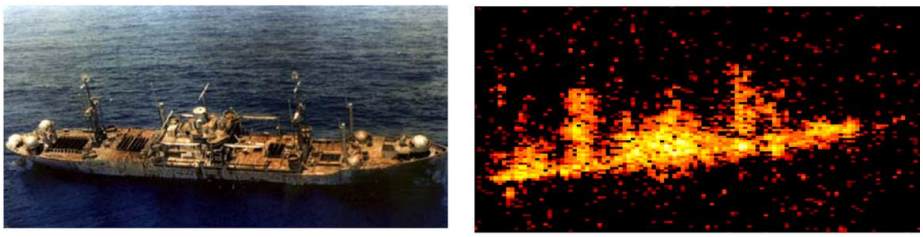
I would like to thank my thesis advisor, Professor Phillip E. Pace, for his guidance over the past year. His patience and eagerness in imparting his wealth of knowledge to everyone is indeed inspiring. I would also like to thank my co-advisor, Professor Douglas J. Fouts, for all his valuable advice. Together, they have made this thesis a truly unforgettable journey. I would also like to express my gratitude to Professor David C. Jenn and Doctor Sebastian Teich, as I have also benefited from their expertise during the course of writing this thesis. The support for this research was made possible by the NPS CRUSER. Many thanks go to Dr. Raymond Buettner. Last, but not least, I would like to thank the Singapore Armed Forces, as well as my former commander, ME6 (NS) Jeffrey Sim, for giving me this opportunity to pursue my postgraduate studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. INVERSE SYNTHETIC APERTURE RADAR

An inverse synthetic aperture radar (ISAR) is an imaging sensor that creates a high-resolution, two-dimensional image of a moving object. An ISAR transmits electromagnetic (EM) radio frequency (RF) waveforms in the direction of the moving object and receives the waveforms that are reflected back. When there is relative motion between the ISAR and the moving object, also referred to as a target, the frequency of the received RF waveform is different from that of the transmitted RF waveform. This change in frequency, also known as a Doppler shift, is a pulse-to-pulse phase rotation and varies according to the range rate of the target. By processing the Doppler returns, we can use an ISAR to create images in the range and cross-range domain [1]. An image of the U.S.S. *Crockett* and an ISAR image of the ship are shown in Figure 1. By observing the two images, we can identify the masts and superstructure of the ship within the ISAR image.



(a) (b)

Figure 1. a) Photo of the U.S.S. *Crockett*, b) ISAR Image of the U.S.S. *Crockett*. Source: [1].

ISAR technology is employed in civilian applications, such as search and rescue operations and space research. In the military, ISAR such as the AN/APS-137B(V) radar is used to conduct maritime surveillance operations to identify, recognize, and classify surface and airborne targets. With the ISAR being able to provide high-quality data such as range, bearing, and position, a vessel patrolling in disputed waters can easily be

detected and identified by an adversary. As ISAR technology matures and ownership becomes more accessible to all, the need for measures to counter ISAR in the electronic warfare (EW) spectrum will increase.

B. COUNTERING THE ISAR

The main techniques of electronic jamming that can be used to disrupt detection by an ISAR are presented in [2]. The first technique uses a noise signal to mask the backscatter return of the target. Although noise jamming can be achieved using low power, its effectiveness is reduced if the jammer has to spread the power over multiple frequencies at the same time. Increasing the power or using additional jammers can address this constraint on power, but the stronger emission increases the target RF signature and can also disrupt its own communication system. The second technique is deceptive jamming where the target replicates a backscatter return waveform that causes the adversary radar to detect a different target. The intent is to mislead the adversary into mistaking the target for a non-hostile vessel in hopes to disrupt an offensive action against it. The advantage of this technique is a lower emission signature that promises higher EM stealth. The disadvantage is the complexity associated with this technique as the jammer must be able to modulate the pulses of the ISAR waveform with the appropriate time delay, phase rotation, Doppler shift, and amplitude such that the false target looks convincing to the adversary [3].

C. DIGITAL IMAGING SYNTHESIZER (DIS)

The DIS was invented by a team of staff and students from the Naval Postgraduate School (NPS) in 2001 to generate image decoys against an ISAR [4], [5], [6]. The DIS is capable of modulating the phase and amplitude of the ISAR waveforms to resemble the desired motion profile and the radar-cross section (RCS) characteristics of the intended false target that appear on the ISAR image. The ISAR waveforms are provided to the DIS from a high-speed digital RF memory (DRFM) [7], which captures and stores the intercepted waveforms. A phase converter is used to extract the phase samples from the digital signals. The phase samples are provided to a finite impulse response arrangement of pipelined range bin Doppler processors with finite resolution. A

prototype version of the DIS was implemented on an Application Specific Integrated Circuit (ASIC), and functional testing was completed in 2007 [8]. In 2015, the NPS Center for Joint Services Electronic Warfare began work to improve the DIS using Field Programmable Gate Arrays.

As the DIS can provide electronic protection (EP) for surface vessels, it is regarded as a threat to ISAR. This view has inspired several ideas for electronic attack against the DIS, and they are summarized in [9]. One of the proposed electronic attack techniques [10] exploits the presence of banding gaps that the false target creates on the ISAR image, which become more significant as the Doppler resolution of the ISAR increases. In addition, the false target lacks the proper backscatter returns such as the ones from the surface of the sea (sea clutter), which must also appear on the ISAR image. In summary, the characteristics of the false target created by the DIS are limited due to the finite resolution of the range-bin Doppler processors and the lack of a proper sea clutter profile.

D. PRINCIPAL CONTRIBUTIONS

The principal contributions of this research are centered around the development of an efficient DIS phase converter and the creation of a realistic sea clutter profile. The major investigations completed are given below.

1. Phase Converter for the DIS Using the CORDIC Algorithm

A phase converter was designed to generate phase samples for the DIS using the Coordinate Rotation Digital Computer (CORDIC) algorithm. The phase converter receives samples of the ISAR waveforms from a DRFM in complex form comprised of an in-phase (I) component and a quadrature (Q) component. The phase converter calculates the phase of this complex signal as a five-bit binary number. A model of the phase converter was built using MATLAB to study the key parameters, such as the number of iterations of the CORDIC process and the number of fraction bits required for accurate results. The study concluded that the phase converter requires 18 iterations of the CORDIC algorithm to compute accurate phase samples quantized at five bits. After

the parameters were calculated, the phase converter is described using the Verilog hardware description language and the latter tested to work using the ModelSim software.

2. Sea Clutter Profile for the DIS

An existing sea clutter simulation model was modified to generate the proper sea clutter in various sea states and wind directions for a high-resolution airborne radar. The model makes use of an empirical model developed by the Naval Research Laboratory (NRL) to calculate the normalized RCS of the sea clutter. Using the fluctuating power and Doppler generated by two random statistical models, we created a power density spectrum of the sea clutter. A portion of the power spectral density was then extracted to create the sea clutter to collocate beside the false target on the ISAR image.

E. THESIS OUTLINE

In Chapter II, the reader can find a discussion of the fundamental ISAR concepts and signal processing. The purpose of this chapter is to provide a background for understanding ISAR reception and image generation. This background also helps the reader to appreciate Chapter III, which features the DIS architecture and its process. The design of the phase convertor that uses the CORDIC algorithm is discussed in Chapter IV, while the modeling and simulation of the sea clutter is discussed in Chapter V. In this chapter how the sea clutter at various sea states appears on the range-Doppler image is discussed. The thesis is concluded in Chapter VI, in which recommendations for future research on the DIS are also mentioned.

II. SYNTHETIC IMAGING SENSORS

An imaging radar such a synthetic aperture radar (SAR) or ISAR is able to generate images in high resolution. The resolution of a radar is its ability to resolve multiple targets located close to one another, and it is determined by parameters of the radar, such as frequency, coherent processing interval, and the dimensions of the antenna. In the case of the ISAR, the rate of rotational motion of the target about its axis can affect the resolution in the cross-range dimension. The purpose of this chapter is to provide readers a summary of how an imaging radar functions and how image processing is carried out in an ISAR so as to allow the reader to appreciate the DIS process, which is featured in the subsequent chapter.

A. RANGE AND CROSS-RANGE RESOLUTION

The range and cross-range resolutions describe the ability of a radar to resolve targets that are very close to one another. The range and cross-range resolution distances are the minimum separations between two targets in order to be differentiated by the radar. In general, it is harder to achieve high cross-range resolution than it is to achieve high-range resolution. For a non-imaging radar, the range resolution distance is defined as

$$d_r = \frac{c}{2\tau} \tag{2.1}$$

where d_r is the range resolution distance, c is the speed of light, and τ is the transmission pulse width. A shorter pulse width reduces the range resolution distance and increases the resolution. As transmitting a short pulse width to achieve high-range resolution requires a high level of power, which presents a separate set of problems, pulse-compression techniques such as pulse coding or frequency modulation are used to provide the sufficiently large bandwidth that corresponds to a short pulse while keeping the power at a manageable level [4].

The cross-range resolution distance is defined as

$$d_{cr} = \frac{R\lambda}{D} \tag{2.2}$$

where d_{cr} is the cross-range resolution distance, R is the target range, λ is the wavelength, and D is the physical dimension of the antenna. From (2.2), a higher cross-range resolution or a smaller cross-range resolution distance can be achieved by decreasing the RF wavelength or increasing the dimension of the antenna. Real-life constraints such as space limitations and the relatively higher atmospheric attenuation of higher-frequency RF waves, however, prevent radar designers from doing so.

B. SAR CONCEPTS

1. Range and Cross-Range Resolution of SAR

SAR, despite not having a large antenna, is able to achieve high cross-range resolution because it is able to emulate a large antenna during operation. This is achieved by making use of the forward motion of the aircraft that has the SAR as its payload [11]. "Each time a pulse is transmitted, the radar occupies a position a little farther along on the flight path. By pointing a reasonably small antenna out to one side and summing the returns from successive pulses, it is possible to synthesize a very long side looking linear array" [12]. The cross-range resolution distance for a SAR is defined as

$$d_{crSAR} = \frac{R\lambda}{2L} \tag{2.3}$$

where d_{crSAR} is the SAR cross-range distance and L is the flight path length, which is also the length of the synthetic linear array. As [11] has shown, the summing of enough returns such that

$$L = \frac{\lambda}{D}R \tag{2.4}$$

and substituting of (2.4) into (2.3) results in the maximum cross-range resolution as

$$d_{crSAR} = \frac{R\lambda}{2\left(\frac{\lambda}{D}R\right)} = \frac{D}{2}. \tag{2.5}$$

From (2.5), we observe that reducing the dimension of the antenna also reduces the cross-range resolution distance. Yet, there is also a limit to how this should be handled since reducing the size of the antenna also reduces its gain factor, which affects detection range.

2. Doppler Processing

When the aircraft carrying the SAR is in motion, there will be Doppler effect present in the backscatter return, causing a difference in frequency of the transmitted waveforms and the received waveforms. Doppler is determined by the radial velocity and is approximated as

$$f_d = \frac{2v_r}{\lambda} \tag{2.6}$$

where f_d is the Doppler shift and v_r is the radial component of the aircraft velocity along the range dimension. For two targets spaced apart by a distance, their Doppler returns are different at a given time instant. The SAR translates the difference in Doppler frequencies into cross-range separation [11]. A Doppler history of the returns from several evenly spaced points on the ground offset from the radar flight path taken is shown in Figure 2. At any given time, the Doppler from each point has a slight difference, which corresponds to their azimuth separation. A motion compensation technique is used to correct phase error introduced when the aircraft is not flying at a constant velocity.

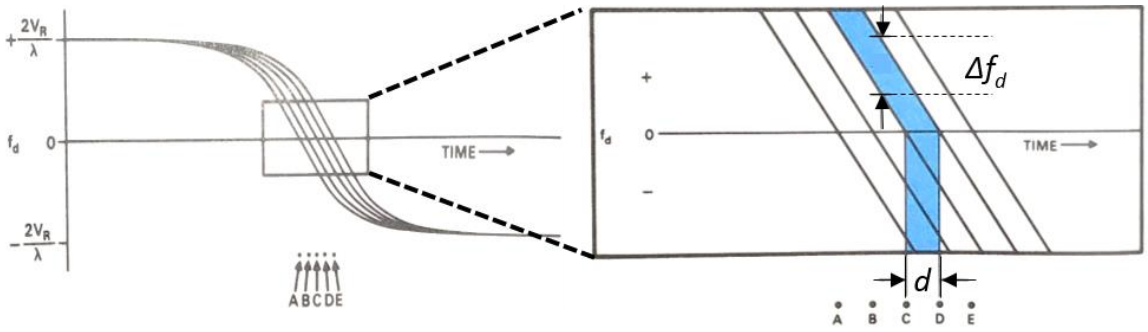


Figure 2. Doppler Histories of Evenly Spaced Points on the Ground.
Source: [11].

3. Spotlight Mode

SAR operates in several modes. These modes are strip-map, squint, and spotlight. Each mode produces an image with different levels of cross-range resolution and are employed for different missions. Spotlight mode offers greater cross-range resolution than the other two modes and is usually employed when there is a region of high interest that requires continuous mapping. When in this mode, the SAR antenna is steered to focus on the region as the aircraft moves around it. Stimson [11] mentioned three improvements using this mode. First, L is no longer bounded by the beamwidth of the physical antenna, which is approximately λ / D. Second, a larger antenna can be used without having to reduce L, thereby bringing about a higher main lobe gain, which improves the signal-to-noise ratio. Finally, viewing the target from different aspect angles allows the SAR to capture returns from other scatterers of the target, which is not achievable from just one aspect angle. The spotlight mode is highlighted in the thesis because it shares the same principle as the ISAR.

C. ISAR CONCEPTS

SAR is suitable for imaging objects when they are stationary. Used on objects that have motions about their own axes, SAR produces images that become blurred. ISAR, which is stationary, makes use of the Doppler shift generated by the object's velocity in the line-of-sight (LOS) direction to the ISAR to resolve the object in the cross-range dimension. A pulse compression technique is used to generate a pulse that has a short width τ and a wide modulation bandwidth Δ with $\tau = 1/\Delta$. A commonly used waveform is the linear frequency modulated (LFM) waveform, which is also known as a chirp waveform. An LFM ISAR pulse can be expressed as

$$r(t) = rect\left(\frac{t}{T}\right)e^{j2\pi\left(f_c+\Delta t^2/2T\right)} \tag{2.7}$$

where T is the uncompressed pulse width in *s*, Δ is the modulation bandwidth of the LFM waveform in Hz, and f_c is the transmitted carrier frequency in Hz. The pulse

envelope of a rectangular function $rect\left(t / T\right)$, the instantaneous modulating frequency, and the resulting LFM pulses are depicted in Figure 3.

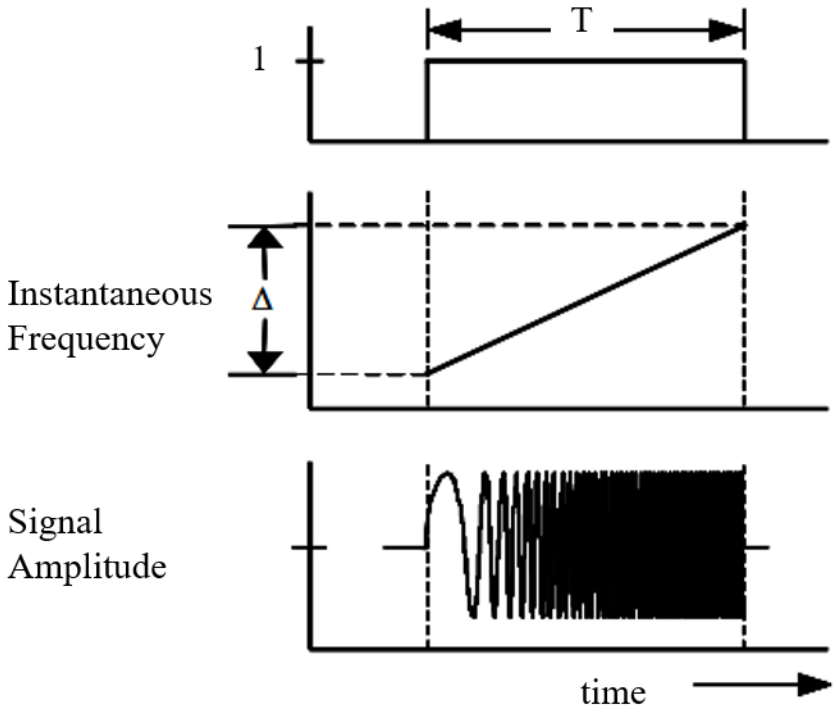


Figure 3. LFM Chirp Pulse. Source: [4].

1. Range Resolution

The range resolution for an ISAR using an LFM waveform is

$$d_{rISAR} = \frac{c}{2\Delta} \ . \tag{2.8}$$

With pulse compression, the improvement in range resolution is a factor of T / τ.

2. Cross-Range Resolution

The cross-range resolution of an ISAR is illustrated in Figure 4. A ship is rolling on its axis perpendicular to the LOS from the ISAR at an angular rate of ω. Point *M*,

which is at the top of the ship mast, is at a distance h away from the axis, giving a tangential velocity of ωh. The instantaneous Doppler frequency shift at that point is

$$f = \frac{2v}{\lambda} = \frac{2\omega h}{\lambda} \ . \tag{2.9}$$

Rearranging gives

$$h = \frac{\lambda}{2\omega} f \ . \tag{2.10}$$

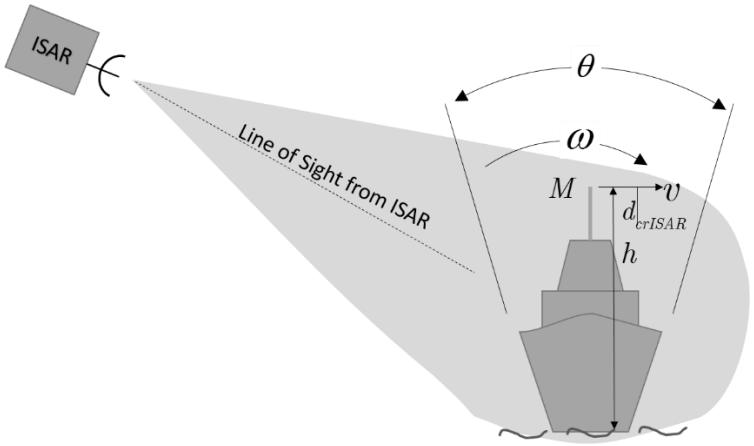


Figure 4. Resolving Cross-Range Resolution Based on Target Rotation

By replacing h with the incremental cross-range distance, which is the cross-range resolution distance of the ISAR d_{crISAR}, we re-express (2.9) as

$$d_{crISAR} = \frac{\lambda}{2\omega} \Delta f \tag{2.11}$$

where Δf is the Doppler resolution of the ISAR. This can also be expressed as

$$\Delta f = \frac{1}{T_I} \tag{2.12}$$

where T_I is the coherent integration time. Substituting (2.9) into (2.8), we can express d_{crISAR} as

$$d_{crISAR} = \left(\frac{\lambda}{2\omega}\right)\left(\frac{1}{T_I}\right) = \frac{\lambda}{2\theta} \tag{2.13}$$

where $\theta = \omega T$ is the angular displacement. From (2.11) and (2.12), we see that d_{crISAR} can be further reduced by increasing the angular velocity, or integration time.

3. ISAR Range-Doppler Image

The ISAR displays a range-Doppler image, which is a consolidation of all the backscatter returns from the target as it rotates on its own axis in a given time period. The range reveals the distance from the ISAR to the target along the ISAR LOS. The Doppler of the target on a plane perpendicular to the ISAR LOS tells the cross-range information. With the ship in Figure 4 as an example, the position coordinates of the mast in the range-Doppler image comprise its position in the range dimension and its Doppler value in the Doppler dimension. When the ship is rolling in the direction of the ISAR LOS, the Doppler shifts created by all the reflective surfaces of the ship are positive. Being the highest located structure on the ship, the mast has the greatest radial velocity and creates the largest positive Doppler shift. When rolling away from the ISAR, the ship appears upside down and the mast has the most negative Doppler value. Such proportionality allows the range-Doppler image to be used interchangeably as a two-dimensional image.

D. ISAR IMAGE PROCESSING

Having introduced the ISAR range-Doppler image and the information that it represents, in this section we discuss the generation of such an image.

1. Range Imaging Model

Consider the scatterers model shown in Figure 5. The range-profile function represents the reflectivity of the target at each resolvable range. For a one-dimensional target with N_r scatterers, the range-profile function can be represented as [4]

$$C_r(x) = \sum_{m}^{N_r} \sigma_m \delta(x - x_m) \tag{2.14}$$

where σ_m is the RCS of the m^{th} scatterer located at range x_m. Let a received ISAR baseband signal from the target scatterer be

$$s(t) = rect\left(\frac{t}{T}\right)e^{j\pi\left(\Delta t^2/T\right)}. \tag{2.15}$$

The received signal can be represented as

$$I(mT_s) = \sum_{m=1}^{N_r}\sigma_m s(mT_s - 2x_m/c) \tag{2.16}$$

where T_s is the analog-to-digital converter (ADC) sampling period and $2x_m/c$ is the round-trip time delay. The received signal from (2.15) can also be expressed as the convolution between the range profile function and the transmitted signal

$$I(mT_s) = C_r(x) \bullet s(mT_s) = C_r(\frac{cmT_s}{2}) \bullet s(mT_s) \tag{2.17}$$

where "$\bullet$" implies convolution. Rearranging the terms in (2.17) and applying Fourier transforms, we get the range profile function, which can be expressed as

$$C_r\left(\frac{cmT_s}{2}\right) = \mathbb{F}^{-1}\left\{\frac{\mathbb{F}\left\{I(mT_s)\right\}}{\mathbb{F}\left\{s(mT_s)\right\}}\right\} = \mathbb{F}^{-1}\left\{\frac{I(k)}{S(k)}\right\} \tag{2.18}$$

where k is the frequency index, $\mathbb{F}$ and $\mathbb{F}^{-1}$ represent the Fourier transform and inverse Fourier transform, respectively.

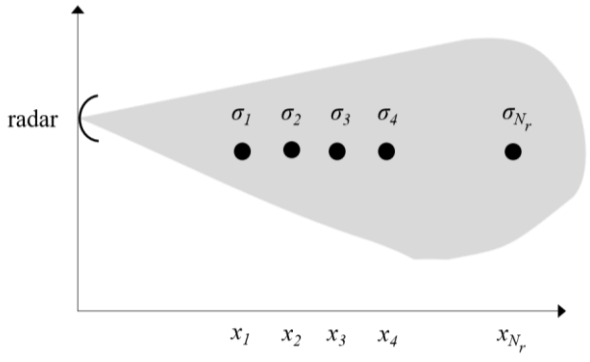


Figure 5. Geometry for One-Dimensional Range Imaging

2. Image Compression

The image compression process is shown in Figure 6. Here *n* is the pulse index and *m* is the sample index of each individual, and *I(m,n)* is the m^{th} sample of the n^{th} pulses received by the ISAR, *S(m,n)* is the m^{th} sample of the n^{th} pulse transmitted, and $C_r(m,n)$ is the range profile of the target based on the m^{th} sample of the n^{th} pulse.

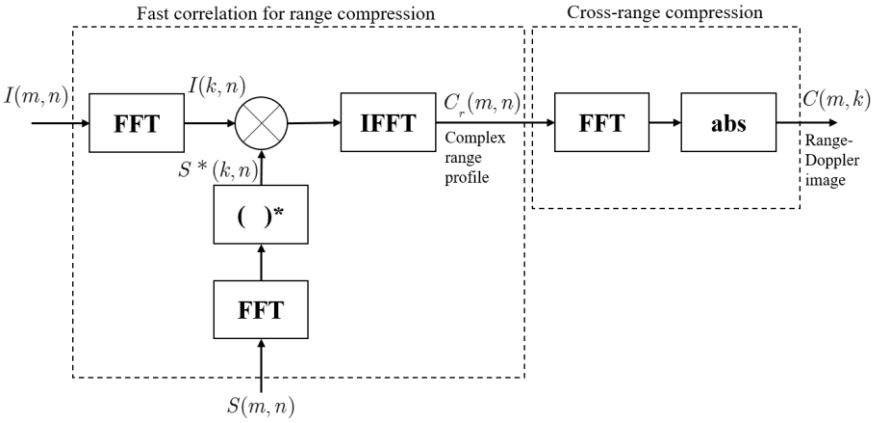


Figure 6. ISAR Compression Process. Source [1].

The inverse Fourier transform in (2.18) assumes *S(k)* is nonzero at all frequencies. In practice, range imaging is based on matched filter theory, and the range profile function is obtained from the cross-correlation of the received signal with the transmitted signal [4]. The new expression for the range profile function is, therefore,

$$C_r(m,n) = I(m,n) \otimes S(m,n) \tag{2.19}$$

where "$\otimes$" implies cross correlation. Putting the cross correlation of *I(m,n)* and *S(m,n)* through an inverse Fourier transform results in the product of the Fourier transform of *I(m,n)* and the complex conjugate of the Fourier transform of *S(m,n)*

$$\mathbb{F}\left\{C_r(m,n)\right\} = \mathbb{F}\left\{I(m,n) \otimes S(m,n)\right\} = I(k,n)S^*(k,n), \tag{2.20}$$

where "*" implies complex conjugate and

$$S^*(k,n) = \left[\mathbb{F}\left\{S(m,n)\right\}\right]^* \tag{2.21}$$

and

$$I(k,n) = \mathbb{F}\left\{I(m,n)\right\}. \tag{2.22}$$

The range profile function can then be obtained by taking the inverse Fourier transform of (2.18), which is

$$C_r(m,n) = \mathbb{F}^{-1}\left\{I(k,n)S^*(k,n)\right\}. \tag{2.23}$$

Equation (2.23) can be displayed as a matrix in which each row is the range profile function of a distinct pulse, and each column holds the component of the range profile grouped into the same range bin. This matrix can be expressed as

$$C_r(m,n) = \begin{bmatrix} C_{rN_p}(m,n) & for\ pulse\ N_p \\ C_{rN_p-1}(m,n) & for\ pulse\ N_p - 1 \\ \vdots & \\ C_{r2}(m,n) & for\ pulse\ 2 \\ C_{r1}(m,n) & for\ pulse\ 1 \end{bmatrix}. \tag{2.24}$$

Every range profile function represents the reflectivity of the target in the time domain. Performing a Fourier transform on the range profile matrix and taking the absolute values yields the Doppler components at each range bin for all the N_p pulses. Aligning the range Doppler for all the pulses then yields the final ISAR image. The ISAR image can be expressed as [1]

$$C_r(m,k) = FFT\{C_r(m,n)\} = \left|\frac{1}{N_p}\sum_{n=0}^{N_p-1} C_r(m,n)e^{-jwT_skn}\right|. \tag{2.25}$$

E. CHAPTER SUMMARY

The concept of SAR and ISAR were explained in this chapter. The processes of range modeling and range compression leading to the generation of the range-Doppler profile were also discussed. An overview of the DIS process is presented in the next chapter.

III. THE DIS PROCESS

The processing of a backscatter signal to form a range-Doppler image within an ISAR was discussed in the previous chapter. In this chapter, the discussion focuses on the DIS and its capability to synthesize a false target in the ISAR image. That is, the DIS modulates the ISAR return waveforms to include imitations of radar return signals such that a false target appears in the ISAR compression.

A. OVERVIEW

At the system level, the DIS is integrated into a DRFM [9]. The block diagram of a DRFM is shown in Figure 7. A DRFM is able to intercept and store RF waveforms as well as retransmit them subsequently. Upon capturing an ISAR waveform, the DRFM uses a local oscillator to down convert the signal to an intermediate frequency. The intermediate frequency signals are separated into in-phase and quadrature components and are digitized by the ADCs into digital samples that are stored in a high-speed memory. An I/Q phase converter extracts phase information from the digitized waveforms to generate phase samples for the DIS to process. After modulation by the DIS is complete, the DRFM converts the processed signal back into an analog form. Finally, the DRFM transmits the analog signal back to the ISAR.

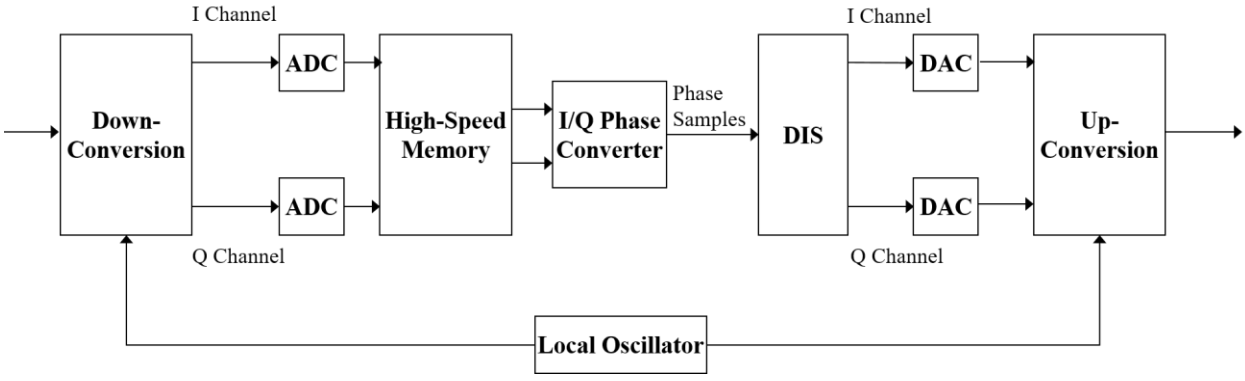


Figure 7. Simplified Block Diagram of the DRFM Integrated with the I/Q Phase Converter and DIS. Adapted from [9].

The range resolution distance of a typical modern ISAR is approximately 1 m. The range resolution of the DIS has to be larger than that of the ISAR in order to have the generated false target look genuine [1]. A target profile of a large structure, such as a ship, spans many range bins. A simplified example used as a test pattern is shown in Figure 8, where the scatterers of a ship target are distributed over 32 range bins. The DIS consists of multiple range bin modulators in parallel with each range bin modulator responsible for modifying one range bin of the target. Unlike the ISAR where the range resolution distance is determined by the bandwidth of the compressed LFM pulse, the range resolution of the DIS depends on the DRFM sampling period or delay between each range bin modulator. Having more range bin modulators gives the DIS the options to synthesize extended targets in range-Doppler space.

The test target in Figure 8 was used to verify that the complex range bin processors were functioning properly. In addition, the test pattern was used to confirm the compression signal processing of the ISAR. The test pattern can be compared to the output range-Doppler configuration to identify problems with the image generation architecture. Most importantly, the test pattern process leads to engineering concepts that must be addressed in the deployed version of the DRFM where realistic targets are synthesized. Engineering details such as clutter modeling and addition of the clutter coefficients to the target coefficients can be studied. The development of realistic target models can also be examined.

B. PHASE AND GAIN COEFFICIENTS

The modulated backscatter returns must have the appropriate amplitudes and phases in order to generate a false target that looks convincing to the ISAR operator. These returns are supposed to resemble the reflected RF waveforms from each scatterer on the target, such as the ones shown in Figure 8. The amplitude and phase of the return signal are determined by the RCS and Doppler frequency of each scatterer, respectively. The false target modulation is represented by a gain coefficient and a phase coefficient. To generate a specific test target profile, the DIS needs to be programmed with a set of gain coefficients and phase coefficients.

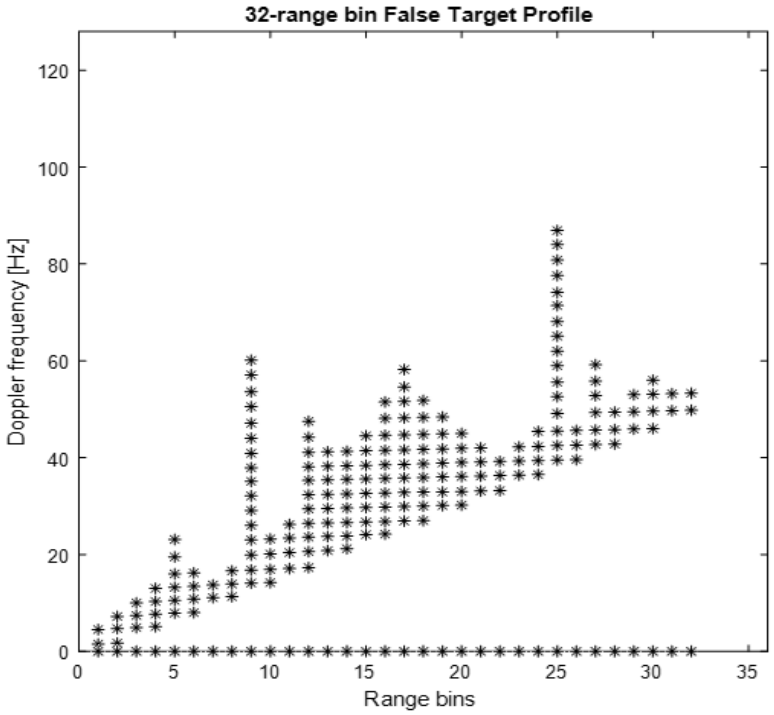


Figure 8. Scatterer Distribution of a Ship Test Target. Source: [5].

1. Generating Realistic Phase and Gain Coefficients

The process for generating phase and gain coefficients to generate a realistic target image is shown in Figure 9. Here, an ISAR is shown illuminating complex scatterers consisting of a ship rocking back and forth in the sea. This complex scattering scene cannot be created synthetically; it can only be generated using the scattered EM field from the ISAR sensor as shown. The scattered waveform is collected by the reference receiver that digitized the scattered field using a high sampling rate. The I and Q components are digitized and used to calculate the gain and phase increment coefficients. It is difficult and expensive to collect the scattered waveforms experimentally. Consequently, detailed models of the ISAR, ship test target, and sea clutter can be used to collect the scattered returns instead. For the purpose of illustrating the bit-level simulation of the DIS, the generation of the coefficients was based on the ship target as a test pattern for the rest of this chapter.

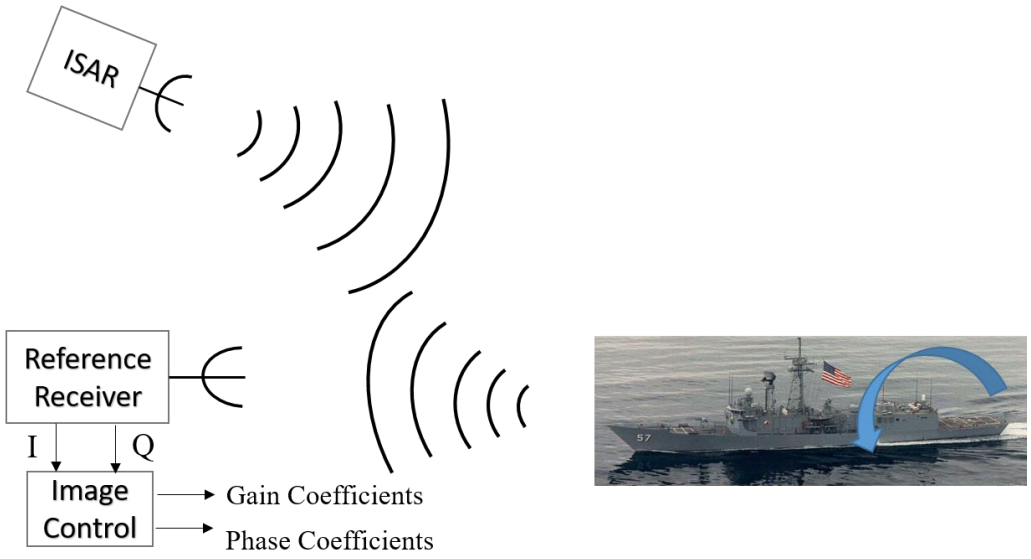


Figure 9. Generating Realistic Phase and Gain Coefficients

2. Generating a Target Return for a Test Target

The phase coefficients were derived using the range-Doppler profile of the test target. An example of a range-Doppler profile of a ship target as a test pattern is shown in Figure 10. Here, *r* is the index of the range cell in the horizontal axis, and *d* is the index of the Doppler cell in the vertical axis [5]. The number that appears in a cell with coordinate *(r,d)* indicates the Doppler of the scatterer in that location. The information about the RCS of the scatterer is not displayed in this range-Doppler profile.

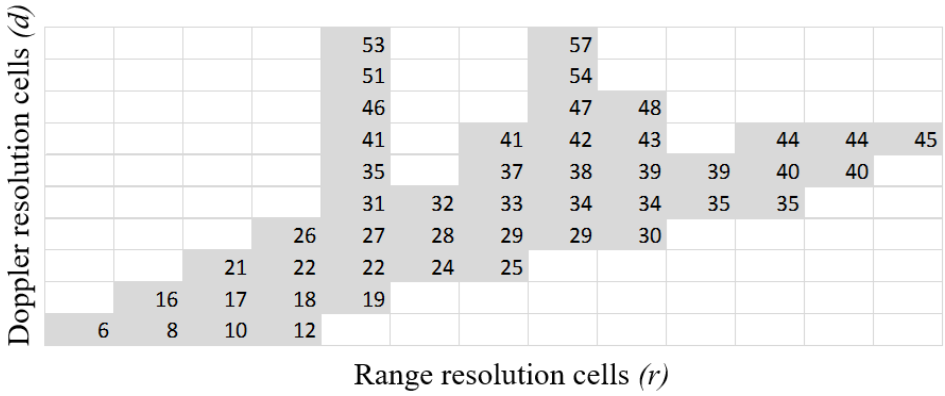


Figure 10. Range-Doppler Profile for a Test Target Template. Adapted from [4].

The ISAR integrates multiple pulses to form an image. Since the target velocity component in the radar LOS and the target position vary with each pulse, the amplitude and phase also vary for different pulses. As such, the target return in the complex form can be represented as a function of the transmitted pulse index and coordinate location on the range-Doppler profile map

$$T(r,d,n) = A(r,d)e^{-j2\pi f(r,d)nPRI} . \tag{3.1}$$

The target return can be consolidated at each range bin for each pulse. This is achieved by adding up the returns from all the scatterers in the Doppler cells with the same range cell index and can be expressed as [5]

$$T'(r,n) = \sum_{d=1}^{N_d} T(r,d,n), \tag{3.2}$$

where N_d is the total number of Doppler cells. Equation (3.2) was used to generate the phase coefficient and gain coefficients.

3. Generating the Phase Coefficients for a Test Target

The phase angles of the target returns at each range bin *r* for each pulse *n* were calculated to derive the phase coefficients. The phase angle of a target return was computed using its real and imaginary components and is expressed as [5]

$$\phi_T(r,n) = \measuredangle\left(\frac{imag\{T'(r,n)\}}{real\{T'(r,n)\}}\right). \tag{3.3}$$

The use of the phase angle is explained in [1]. "To create the proper Doppler frequency, the image synthesizer rotates the DFRM phase samples $\phi(m,n)$ on a pulse-to-pulse basis. Therefore, instead of the extracted phase value being applied directly, a phase increment is required" [1]. The phase increment for range bin *r* and pulse *n* can be expressed as

$$\phi'_{inc}(r,n) = \phi'_{inc}(r,n-1) + \phi_T(r,n-1) - \phi_T(r,n) . \tag{3.4}$$

The phase coefficient is generated when the phase increment is quantized at *kp*-bits, which is

$$\phi_{inc}(r,n) = \left\lfloor \frac{\phi'_{inc}(r,n)2^{k_p}}{2\pi} \right\rfloor \tag{3.5}$$

where $\lfloor \ \rfloor$ is the floor integer rounding function.

4. Generating the Gain Coefficients for a Test Target

From [4], the amplitude modulation involves taking the product of the intercepted signal and "a gain value equal to the magnitude of the sum of the complex scatterers at the same range bin/pulse combination." To obtain the gain values, the magnitude in (3.2) is normalized so that it varies between 0 and 1

$$T'_N(r,n) = \frac{|T'(r,n)|}{\max|T'(r,n)|}. \tag{3.6}$$

The gain values are applied by performing bit-shifting on the I and Q components of $T'(r,n)$. The extent of the shifting is determined by a control code k_g, which is assigned based on the ranges of magnitude for $T'(r,n)$ [5]. As an illustration, the gain modulation quantization scheme that uses four bits for k_g is shown in Table 1. The gain coefficient $g(r,n)$ represents the value of the shift in the binary signal with the effective gain, as shown in Table 1.

Table 1. Gain Modulation Quantization Scheme. Source: [4].

Normalized Magnitude	Gain Coefficient $g(r,n)$	Effective Gain $2^{g(r,n)}$
0.8–1.0	10	1024
0.4–0.8	9	512
0.2–0.4	8	256
0.1–0.2	7	126
0.05–0.1	6	64
0.025–0.05	5	32
0.0125–0.025	4	16
0.00625–0.0125	3	8
0.0032–0.00625	2	4
0.0016–0.00625	1	2
0–0.0016	0	1

C. PHASE SAMPLES OF INTERCEPTED ISAR SIGNAL

The DRFM down converts the ISAR received LFM waveforms. Recall the complex envelope of the signal after down conversion is

$$s(t) = rect\left(\frac{t}{T}\right)e^{j\pi(\Delta t^2/T)}. \tag{3.7}$$

Samples of the radar signal are taken by the DRFM at regular intervals. The DRFM sample times are defined as

$$t = \frac{m}{f_s} + nPRI \tag{3.8}$$

where f_s is the ADC sampling frequency and *PRI* is the ISAR pulse repetition interval in s [5]. After substituting the time t in (3.7) with the expression of t in (3.8), we express the complex envelope amplitude sample for the I and Q components as

$$S_{real}(m,n) = real\left\{S(m,n)\right\} \tag{3.9}$$

and

$$S_{imag}(m,n) = imag\left\{S(m,n)\right\}. \tag{3.10}$$

The I/Q phase converter calculates the phase angles of the I and Q sample signal and generates them as k_p-bits phase samples. The phase angles can be expressed as

$$\phi_o(m,n) = \measuredangle\left(\frac{S_{imag}(m,n)}{S_{real}(m,n)}\right), \tag{3.11}$$

and the phase samples can be expressed as

$$\phi(m,n) = \left\lfloor \frac{\phi_o(m,n)2^{k_p}}{2\pi} \right\rfloor. \tag{3.12}$$

The I/Q phase converter is implemented using the CORDIC algorithm and is discussed in greater detail in Chapter IV.

D. DIS SIGNAL PROCESSING

In this section, we describe the signal processing that takes place within the DIS. The block diagram of the DIS is shown in Figure 11. The most significant components of the DIS are the range bin modulators. Each range bin modulator consists of a phase

adder, a look-up table (LUT), a gain block, and a summation adder. At the adder of a specific range bin modulator, a phase sample from the I/Q phase converter is added with a phase coefficient that is designated for that specific range bin modulator and for that specific pulse. The phase increment in a range bin changes with the pulse-repetition interval (PRI). The modulated k_p-bits phase output can be expressed as

$$\hat{\phi}(r,m,n) = \phi(m,n) + \phi_{inc}(r,n)\,. \tag{3.13}$$

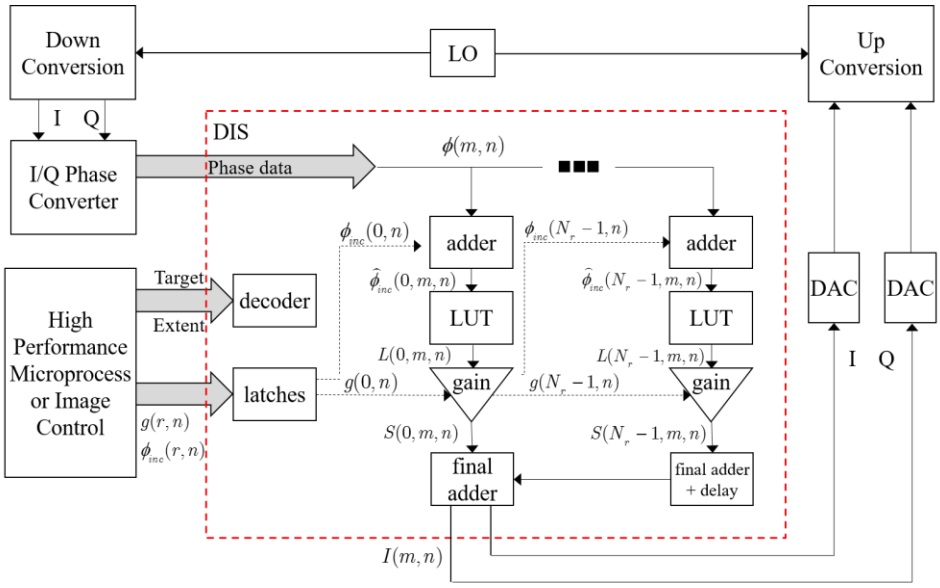


Figure 11. DIS Block Diagram. Adapted from [1].

The modulated phase output is used to construct a complex signal with a unit amplitude that can be expressed as

$$L(r,m,n) = \cos(\hat{\phi}(r,m,n)) + j\sin(\hat{\phi}(r,m,n))\,. \tag{3.14}$$

The cosine and sine of the modulated phase are computed using an LUT that contains the results of the cosine and sine operation on all 2^{k_p} possible phase outputs. With that, the normalized I and Q amplitude components are established and applied to the gain block via their own channel [5].

The amplitude modulation takes place at the gain block where the I and Q components are multiplied with the gain coefficients. The outcome of this amplitude modulation is a complex signal that can be expressed as

$$S(r,m,n) = 2^{g(r,n)} L(r,m,n) = 2^{g(r,n)} e^{j(\phi(m,n)+\hat{\phi}_{inc}(r,n))}. \quad (3.15)$$

The final adder is the last stage in each range bin modulator. The complex signal in (3.15) is sent to the final adder and added with the output of its adjacent final adder from another range bin modulator. For example, the output of the final adder from range bin modulator #1 is the summation of the outcome of the amplitude modulation in range bin modulator #1 and the output of the final adder from range bin modulator #2. The output of the final adder from range bin modulator #2 is the summation of the amplitude modulation created in range bin modulator #2 and the output of the final adder from range bin modulator #3, and so on. The complex output pulse from the DIS is taken from the final adder in range bin modulator #1. A delay of one clock cycle is introduced by each adder. In other words, "each output pulse is the superposition of N_r copies of the pulse, each delayed with respect to another by the adder delay, scaled differently by the gains $2^{g(r,n)}$ and phase rotated by $\phi_{inc}(r,n)$" [1]. The final DIS output expression is given by

$$I(m,n) = \sum_{r=0}^{N_r-1} S(r,m,n) = \sum_{r=0}^{N_r-1} 2^{g(r,n)} e^{j(\phi(m-r,n)+\phi_{inc}(r,n))}. \quad (3.16)$$

The output from the DIS is returned to the DRFM, which converts the output signal into analog form. After that, the analog signal modulates the carrier frequency of the ISAR prior to transmission back to the ISAR. Upon reception of the modulated signal, the ISAR processes the signal using the imaging process described previously in Chapter II.

E. RANGE-DOPPLER IMAGE FOR A TEST TARGET

In this section, we presents the range-Doppler image for a false target that is generated by the DIS using the test target shown in Figure 8. The DIS was previously been implemented in MATLAB. The bit-level simulation algorithm of the software model is covered in detail in [4]. The MATLAB comprises three main files that must be

executed in the correct order. The first file, extract_v5.m, generates the modulation coefficient for a test target. The second file, mathost_v5.m, simulates the phase samples provided by the DRFM and the reference signal required by the ISAR receiver. The last file, simhwchk_v5, simulates the DIS signal processing, generates the modulated ISAR waveforms, and simulates the ISAR compression process to produce a range-Doppler image of the test target. The key parameters used in [5] are consolidated in Table 2.

Several test target profiles with different target extent are also created. The choice of the test target profile decides the number of range bin modulators involved to perform the modulation. The scatterer distribution for a test target spanning 32 range bins as a test pattern is displayed in Figure 8. The simulated range-Doppler image of a test target spanning 32 range bins is displayed in Figure 12. By comparing Figure 8 and Figure 12, we can recognize the superstructure of the ship such as the two masts at both ends of the ship as well as the wheelhouse located in the center of the ship.

Table 2. Key Parameters Used in the Simulation of the DIS

1. ISAR LFM Waveforms	
PRF	200 Hz
Number of pulses for integration	128
Bandwidth	500 MHz
2. DRFM	
Sampling frequency	1 GHz
3. DIS	
Number of bit for phase samples	5 bits
Number of bit for phase quantization, k_p	5 bits
Number of bit for gain control code, k_g	4 bits

F. IDENTIFYING THE DIS TEST TARGET

Compared with the ISAR image of the U.S.S. *Crockett* in Figure 1, there appear to be banding gaps consistently displayed in the Doppler frequency domain of Figure 12. This can be explained by the limitation of the ship target profile. The surface of a real ship is continuous, whereas the ship target profile is constructed using a limited number of discrete scatterers. The latter results in the Doppler component of the target return to be discrete, which causes the gaps to appear. Using the coefficients calculated with the actual scattered electromagnetic field can eliminate the presence of these banding gaps due to the complex interactions of the near-field reflections.

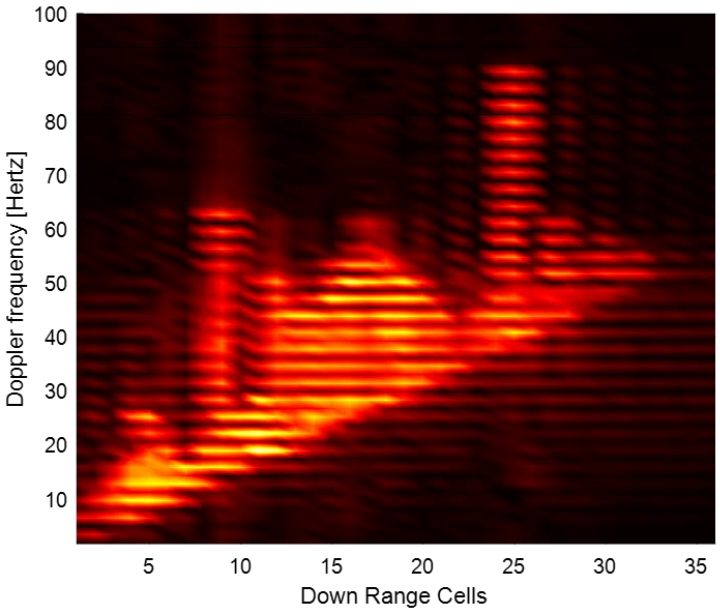
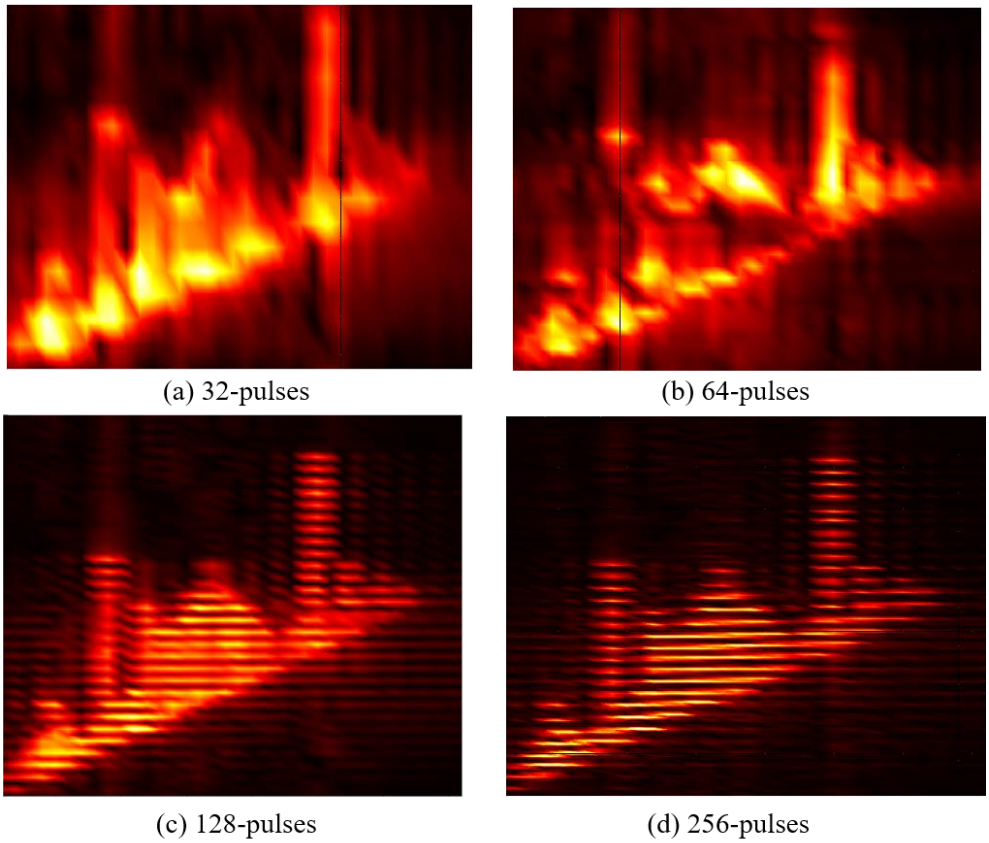


Figure 12. Simulated Range-Doppler Image of 32-Range Bin Test Target

The ISAR image of the U.S.S. *Crockett* in Figure 1 also contains blurred regions, whereas the false target created by the DIS using the test pattern appears clear and distinct. The real image is blurred because the movement of a real ship is complex and cannot be adequately compensated by the ISAR image realignment and phase compensation. The DIS image appears sharp because the false target signal created using digitally quantized phase and gain coefficient can be compensated fully by the ISAR.

The requirement on the fidelity of the range-Doppler image varies according to mission requirements and platform capability. The fidelity can be enhanced by increasing the number of pulses integrated into the ISAR compression process as shown in Figure 6. When that happens, the lack of details on the test target is even more apparent and is demonstrated in Figure 13. The compression results provide significant insight into the coefficient to the DIS image formation. Although the scattered EM field from an actual ship in the sea provides a realistic target signature from which target coefficients can be extracted, no insight into the overflow characteristics and gain quantization effects in the image formation process and the influence of the coefficients on that process can be determined.



(a) 32-pulses (b) 64-pulses

(c) 128-pulses (d) 256-pulses

Figure 13. Gaps Developing on False Target as Number of Integrated Pulses Increases

G. CHAPTER SUMMARY

In Chapter III, an overview of the DIS was provided. Also described was the generation of the modulation coefficients using the target return constructed based on a test target profile. The internal operation of the DIS was also featured in the chapter. A false target image was presented at the end of the chapter, and some of the existing shortcomings of the image were highlighted as well. In the next chapter, the design of the I/Q phase processor that uses the CORDIC algorithm is featured.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. THE CORDIC PROCESSOR

A. OVERVIEW

The I/Q phase converter is a critical component of the DIS electronic warfare architecture as it is responsible for providing the phase samples to the DIS for gain and phase modulation. The origins of the phase samples are the transmitted RF waveforms intercepted and stored in the DRFM. The I/Q phase converter processes the I and Q components of these digitized waveforms and creates phase samples for the DIS.

1. Arctangent Function

The I and Q components of an ISAR waveform sample can be combined to form a complex number z = x + jy, with the I component being the real part and the Q component being the imaginary part. The phase of z is defined as the angle from the positive real x-axis in the counter clockwise direction. A way to measure this angle is to perform the division of y by x and apply the result to the arctangent function. An example of angle measurement for a complex number is shown in Figure 14.

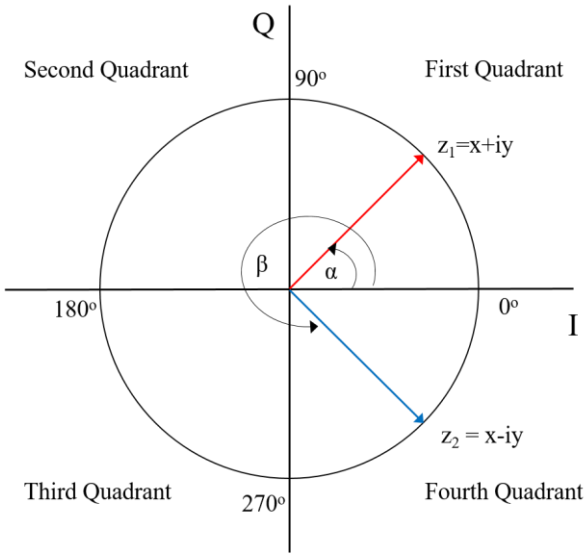


Figure 14. Angular Measurement for a Complex Number

The phase angle of the complex number z_1 is α, which can be expressed as

$$\begin{aligned}\alpha &= \arctan\left(\frac{y}{x}\right) \\ &= \tan^{-1}\left(\frac{y}{x}\right).\end{aligned} \tag{4.1}$$

The phase angle of the complex number z_2 is β,which can be expressed as

$$\beta = \arctan\left(\frac{-y}{x}\right). \tag{4.2}$$

As β is a reflex angle (more than 180^o), it can also be expressed as

$$\beta = 2\pi - \tan^{-1}\left(\frac{y}{x}\right) = 2\pi - \alpha\,. \tag{4.3}$$

2. Phase Quantization

The phase samples and the phase coefficients are quantized at k_p-bits. The current version of the DIS is based on a five-bit quantization scheme where 32 distinct five-bit binary words are used to represent angles from 0^o to 360^o. The angle resolution is, therefore, 11.25^o per bit. The representation of the phase angles using five-bit phase resolution is displayed in Table 3.

B. COORDINATE ROTATION DIGITAL COMPUTER (CORDIC)

The CORDIC algorithm is chosen as the method to perform the complex-to-phase conversion. The algorithm uses vector rotations to solve trigonometric functions. There are two main reasons for choosing CORDIC. First, it is a hardware efficient algorithm that implements vector rotations using only shift and add operations. Second, it allows a pipelined architecture, which optimizes the throughput.

Table 3. Representation of Phase Angles Using Five-Bit Resolution

Binary Representation	Values of Binary Representation	Phase Angle	Binary Representation	Values of Binary Representation	Phase Angle
00000	0	0	10000	16	180
00001	1	11.25	10001	17	191.25
00010	2	22.5	10010	18	202.5
00011	3	33.75	10011	19	213.75
00100	4	45	10100	20	225
00101	5	56.25	10101	21	236.25
00110	6	67.5	10110	22	247.5
00111	7	78.75	10111	23	258.75
01000	8	90	11000	24	270
01001	9	101.25	11001	25	281.25
01010	10	112.5	11010	26	292.5
01011	11	123.75	11011	27	303.75
01100	12	135	11100	28	315
01101	13	146.25	11101	29	326.25
01110	14	157.5	11110	30	337.5
01111	15	168.75	11111	31	348.75

1. CORDIC Theory

The CORDIC theory is covered in this section. In Figure 15 a vector with magnitude Z has a phase angle of θ. It is subsequently rotated by an angle φ in the counter clockwise direction. At its first position, the components of the vector can be expressed as

$$x_o = Z\cos\theta \tag{4.4}$$

and

$$y_o = Z\sin\theta. \tag{4.5}$$

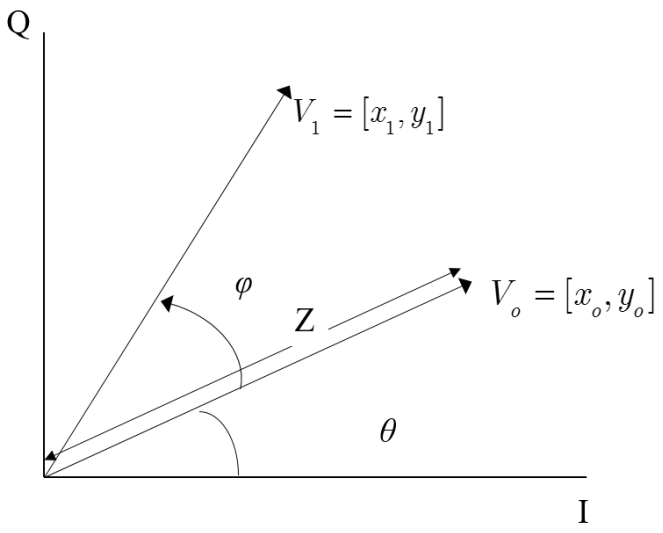


Figure 15. A Vector Being Rotated Counter-Clockwise by Angle φ

After the rotation, the components of the resulting vector can be expressed as

$$\begin{aligned} x_1 &= Z\cos(\theta+\varphi) \\ &= Z\left(\cos\theta\cos\varphi - \sin\theta\sin\varphi\right) \\ &= Z\cos\theta\cos\varphi - Z\sin\theta\sin\varphi \\ &= x_o\cos\varphi - y_o\sin\varphi \\ &= \cos\varphi(x_o - y_o\tan\varphi) \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} y_1 &= Z\sin(\varphi+\theta) \\ &= Z\left(\cos\theta\sin\varphi + \sin\theta\cos\varphi\right) \\ &= Z\cos\theta\sin\varphi + Z\sin\theta\cos\varphi \\ &= x_o\sin\varphi + y_o\cos\varphi \\ &= \cos\varphi(y_o + x_o\tan\varphi). \end{aligned} \tag{4.7}$$

By conforming $\tan\varphi$ to take on values such as $\pm 2^{-i}$ where i is the rotation index, the multiplication of $\pm 2^{-i}$ is equivalent to a shift operation that can be easily implemented. The $\cos\varphi$ term can be expressed

$$\begin{aligned} \cos\varphi &= \cos\left(\tan^{-1}\left(2^{-i}\right)\right) \\ &= \frac{1}{\sqrt{1+2^{-2i}}}. \end{aligned} \tag{4.8}$$

Being an even function, (4.8) can be treated as a scale constant regardless of the direction of rotation. The general expression for the x and y components of the resulting vector after every subsequent rotation are x_i and y_i, [12] and can be expressed as

$$x_{i+1} = \frac{1}{\sqrt{1+2^{-2i}}}(x_i - y_i d_i 2^{-i}) \tag{4.9}$$

and

$$y_{i+1} = \frac{1}{\sqrt{1+2^{-2i}}}(y_i + x_i d_i 2^{-i}) \tag{4.10}$$

where $d_i = \pm 1$ depending on the direction of rotation. The constant scale factor for every rotation $1/\sqrt{1+2^{-2i}}$ can be collectively treated as the system gain A_n of the CORDIC processor. This system gain can be expressed as

$$A_n = \prod_n \sqrt{1+2^{-2i}} \ . \tag{4.11}$$

As the number of rotations increases, the system gain approaches a value of 1.647.

With the rotation angle conformed to $\tan^{-1}\left(2^{-i}\right)$, the magnitude becomes smaller with each rotation. An angle accumulator can be used to track the net angular displacement in one direction. If the initial angle is made known as well, the angle accumulator reveals the vector angle after the final rotation. The angle accumulated can be defined as

$$z_{i+1} = z_i - d_i \tan^{-1}\left(2^{-i}\right). \tag{4.12}$$

2. Vector Rotation Mode

There are two operating modes of CORDIC: rotation mode and vectoring mode. In rotation mode, a vector is made to rotate a specified angle in one direction progressively over several smaller rotations. The vectoring mode is used to find the angle of an input vector. At each iteration, the CORDIC rotates the vector toward the positive axis with the objective of reducing the y component of the resulting vector after each rotation. If the y component is negative, the next rotation is made in the counterclockwise direction, and the angle accumulator subtracts the angle displaced from the last rotation. If the y component is positive, the resultant vector is rotated in the clockwise direction,

and the angle accumulator adds the last displaced angle. At the end of the final iteration, the angle accumulator contains the total angular displacement, which is the result of the arctangent function. The CORDIC equations in vectoring mode are

$$x_{i+1} = x_i - y_i d_i 2^{-i}, \tag{4.13}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i}, \tag{4.14}$$

and

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \tag{4.15}$$

where $d_i = +1$ if $y_i < 0$ and $d_i = -1$ if y_i is otherwise. As the number of iterations approaches infinity, the final results are

$$x_n = A_N \sqrt{x_o^2 + y_o^2}, \tag{4.16}$$

$$y_n = 0, \tag{4.17}$$

$$z_n = z_0 + \tan^{-1}\left(\frac{y_0}{x_0}\right), \tag{4.18}$$

and

$$A_N = \prod_n \sqrt{1 + 2^{-2i}}. \tag{4.19}$$

3. Initialization of the Angle Accumulator

The CORDIC algorithm mentioned so far in this chapter is only effective when the angle of the vector of interest is between -90^o and 90^o, a range that lies in the first and fourth quadrants. To include vectors in the other two quadrants, an initial rotation is required to bring the vector into either the first quadrant or the fourth quadrant. In other words, z_0 is initialized at 0^o when the vector is in the first or fourth quadrant, 90^o when it is in the second quadrant, and -90^o when it is in the third quadrant [13]. The values for x_o, y_o, and z_o after the initialization are summarized in Table 4. A flow chart that summarizes the CORDIC vectoring mode is shown in Figure 16.

Table 4. Values of x_o, y_o, and z_o for Different Quadrants

Quadrant 1 & 4	Quadrant 2	Quadrant 3
$x_0 = I$ $y_0 = Q$ $z_0 = 0$	$x_0 = Q$ $y_0 = -I$ $z_0 = \pi/2$	$x_0 = -Q$ $y_0 = I$ $z_0 = -\pi/2$

4. Design Methodology for I/Q Phase Converter

The first step to design the I/Q phase converter is to model a CORDIC processor in MATLAB. The objective is to develop a benchmark to evaluate the performance of the CORDIC processor when it is modeled in the Verilog hardware description language. Although there is an existing CORDIC function built inside MATLAB, building one from scratch grants the design team full control of the CORDIC algorithm and access to x_i, y_i, and z_i, which can subsequently be used to troubleshoot the Verilog model. The MATLAB model was first built using floating-point numbers to allow us to focus on addressing implementation error and algorithmic error. After the implementation of the CORDIC in floating-point format was assessed to be correct, a new model that calculated using fixed-point numbers was subsequently developed.

C. FLOATING-POINT MODEL

The floating-point model is relatively straightforward to implement as it can be constructed based on the flow chart in Figure 16. The inputs for the model are the I and Q values, which represent the vector Z = I + j Q. The number of rotations or iterations is decided by the parameter "*iter*." Since the number of iterations affects the accuracy of the results, the design requires a suitable figure be identified.

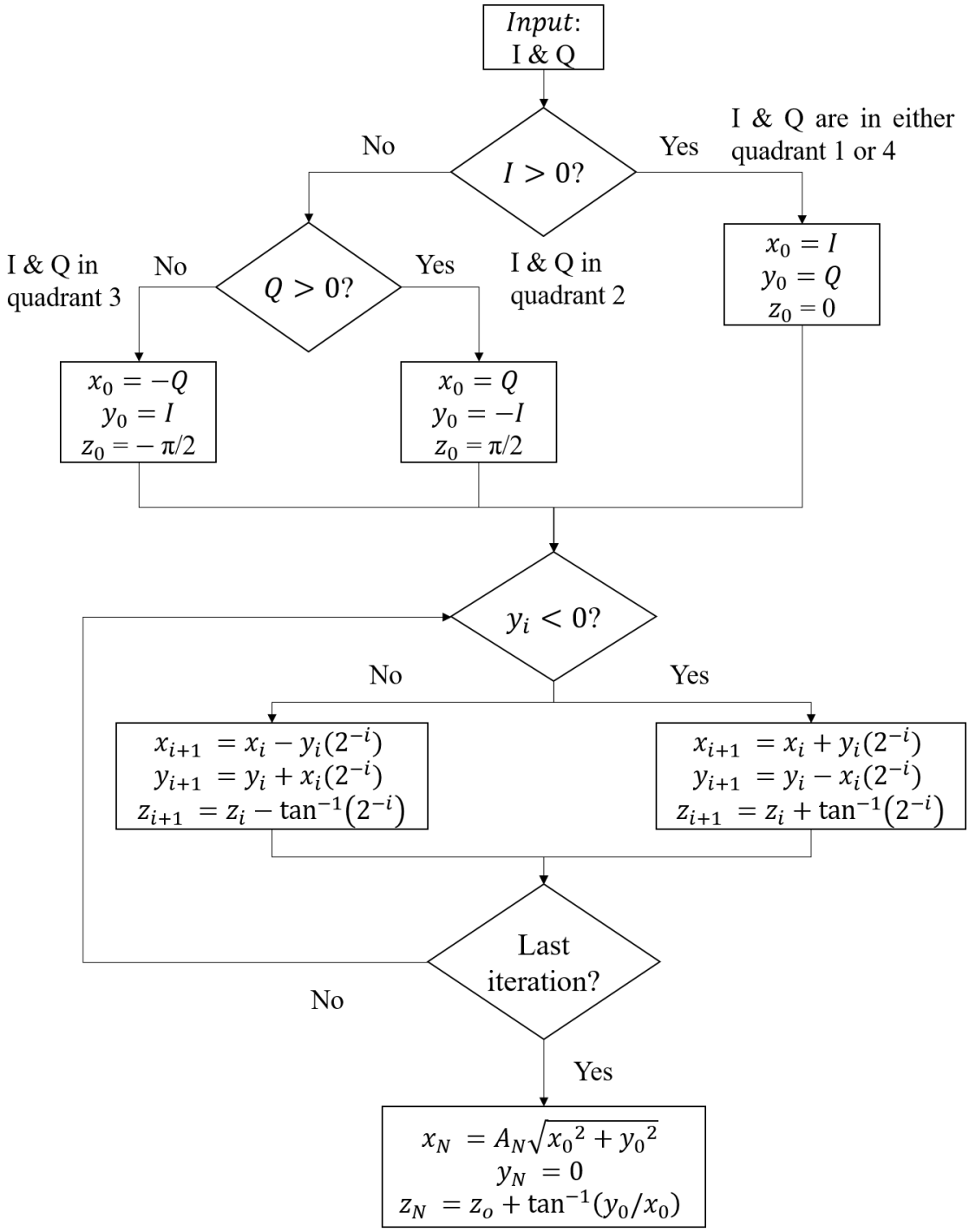


Figure 16. Flow Chart for CORDIC Vectoring Mode

Before commencing the iterative rotation, the model determines whether there is a need for an initial rotation by checking the sign of I and Q to determine in which quadrant the input vector lies. The coordinates and angle of the vector after the initial rotation are stored inside three arrays: "*X*," "*Y*," and "*Z*." The code to execute this check is listed as follows:

```
if (I>=0 && Q >=0) || (I>=0 && Q<0) %Quadrant 1 or 4
    X(1) = I;
    Y(1) = Q;
    Z(1) = 0;
elseif I<0 && Q>=0 % Quadrant 2
    X(1) = Q;
    Y(1) = -I;
    Z(1) = 90;
elseif I<0 && Q<0 % Quadrant 3
    X(1) = -Q;
    Y(1) = I;
    Z(1) = -90;
end
```

The rotation angles applied for each rotation are stored in the array "*angleLUT*," which is defined as:

```
angleLUT = atand(2.^-(0:iter-1));
```

For a CORDIC processor that executes eight rotations, the stored rotation angles are shown in Table 5.

Table 5. Rotation angles for an Eight-Iteration CORDIC Processor

Iteration	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Angle (degree)	45	26.5651	14.0362	7.1250	3.5763	1.7899	0.8952	0.4476

The rotation angles become smaller and more precise after each subsequent iteration, which refines z_i to become closer to the actual phase angle of the vector.

The series of recursive rotations begins next. The direction of rotation "*d(i)*" is determined by the sign of y_i. The code to execute the series of rotations is listed as follows:

```
% rotate the vector for iter-number of times
for i = 1:iter
    if Y(i) < 0
        d(i) = 1;
    else
        d(i) = -1;
    end
    X(i+1) = X(i) - Y(i)*d(i)*2^-(i-1);
    Y(i+1) = Y(i) + X(i)*d(i)*2^-(i-1);
    if(X(i+1)==0 && Y(i+1)==0) %if the inputs are the origin points
       Z(i+1)=0;
    else
        Z(i+1) = Z(i) - d(i)*angleLUT(i);
    end
    fprintf('Iteration: %2d, Calculated angle: %7.3f, Rotated angle: %7.3f, Error in
degrees: %10g, Error in bits: %g\n',...
        [(i); Z(i);  - d(i)*angleLUT(i);(Z(i)-ideal_angle);log2(abs(Z(i)-ideal_angle))]);
end
```

1. Simulation and Results

A vector Z= −45 + j23 with an angle of 152.928^o was used to demonstrate the CORDIC functionality. A visual representation of the vector rotations is presented in Figure 17. The input vector, being in the second quadrant, was rotated 90^o in the clockwise direction to the first quadrant. The series of eight rotations followed next, gradually moving the vector towards the positive X-axis. The accumulated angle z_i after every iteration is shown in Figure 18. It can be seen that z_i tends to converge toward the actual phase angle of the vector.

The numerical results from every iteration of the CORDIC algorithm are presented in Table 6. The angular displacement introduced by each rotation is listed in the column titled “Rotated Angle.” The absolute values of these angles equal the angles kept in “*angleLUT*”, and their signs indicate the direction of rotation. In the “Absolute phase error” column, the difference between the phase angles of the vector Z and z_i after every rotation is shown, and they tend to get smaller, meaning that z_i in general gets closer to the actual phase value as well.

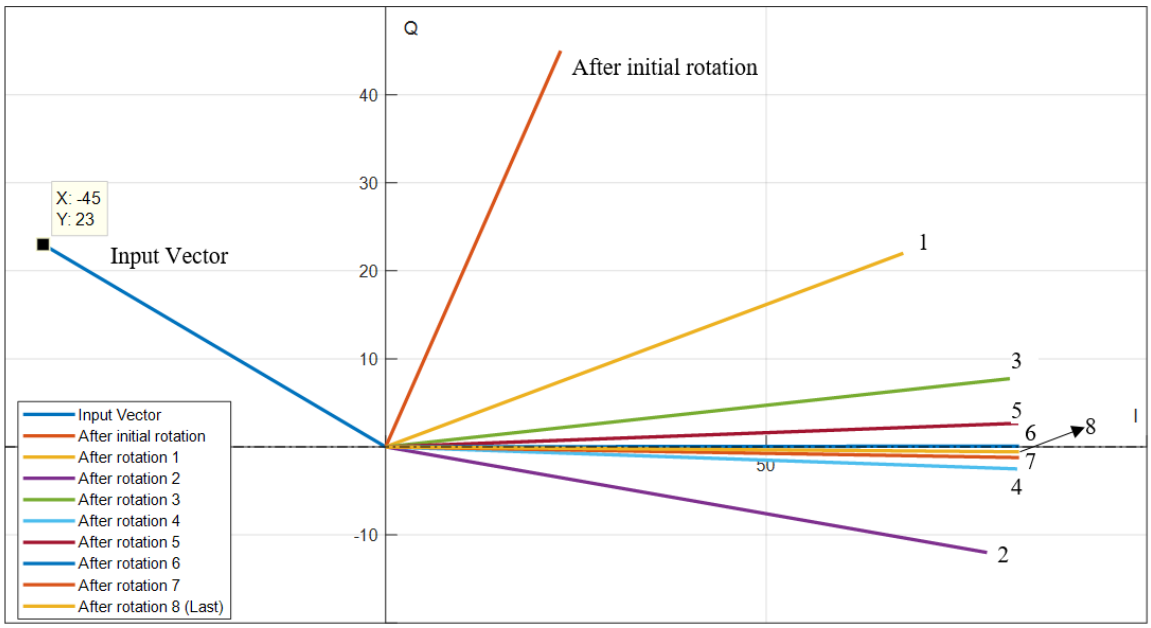


Figure 17. Vectoring Mode CORDIC Iterations

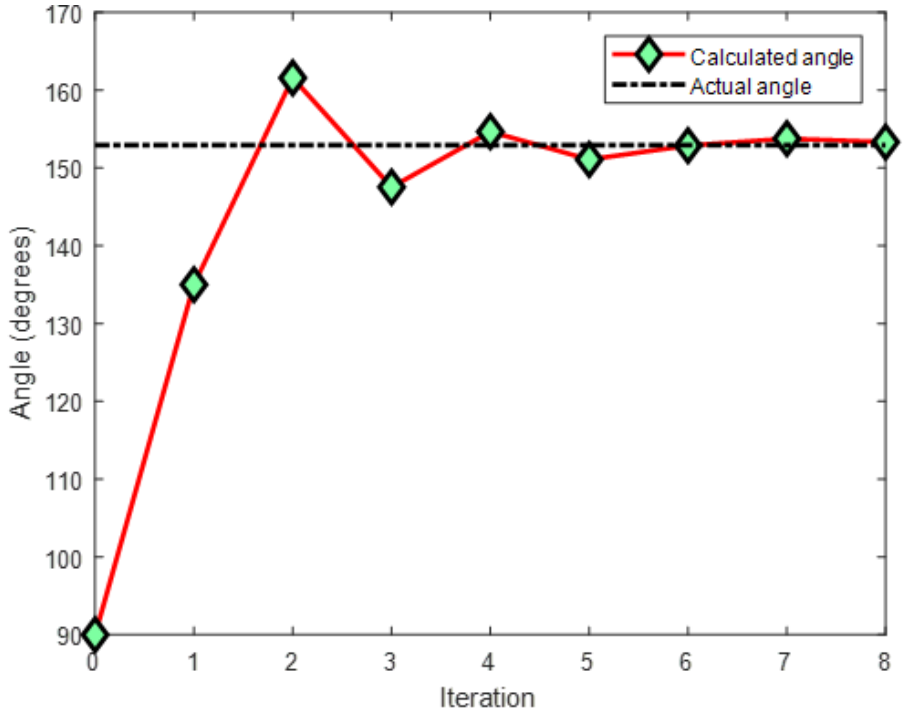


Figure 18. Cumulative Angle through Iterations

Table 6. CORDIC Calculation for Each Iteration

Iteration	x_i	y_i	z_i (deg)	Rotated Angle (deg)	Absolute phase error (deg)
i=0	23	45	90	NA	62.92791976
i=1	68	22	135	-45	17.92791976
i=2	79	-12	161.5650512	-26.56505118	8.637131415
i=3	82	7.75	147.5288077	14.03624347	5.399112053
i=4	82.96875	-2.5	154.6538241	-7.125016349	1.725904296
i=5	83.125	2.685546875	151.0774897	3.576334375	1.850430079
i=6	83.20892334	0.087890625	152.8674003	-1.789910608	0.060519471
i=7	83.21029663	-1.212248802	153.762574	-0.89517371	0.83465424
i=8	83.21976732	-0.56216836	153.3149598	0.447614171	0.387040069

The magnitude of Z is $\sqrt{45^2 + 23^2} = 50.5371$. The magnitude calculated by the CORDIC algorithm is actually the magnitude of x_i. In Table 7, we see that as the iteration continues, the magnitude of x_i has a scale factor that tends toward 1.647.

Table 7. Magnitude of x_i and Scale Factor in Each Iteration

Iteration	X_i	$\lvert -45+23j \rvert / X_i$
i=0	50.53711507	1
i=1	71.47027354	1.414213562
i=2	79.906195	1.58113883
i=3	82.36542054	1.629800601
i=4	83.00640624	1.642484066
i=5	83.16837011	1.645688916
i=6	83.20896976	1.646492279
i=7	83.21912648	1.646693254
i=8	83.22166609	1.646743507

If I and Q are eight-bit binary words, there are a total of 261,121 combinations of inputs in all four quadrants. The surf plots of the results of the arctangent(Q/I) using a trigonometric function and using an eight-iteration CORDIC are presented in Figure 19 and Figure 20. The two surf plots bear a strong resemblance to each other. Based on the results obtained, we assessed that the CORDIC precise calculation was implemented correctly in the floating-point format.

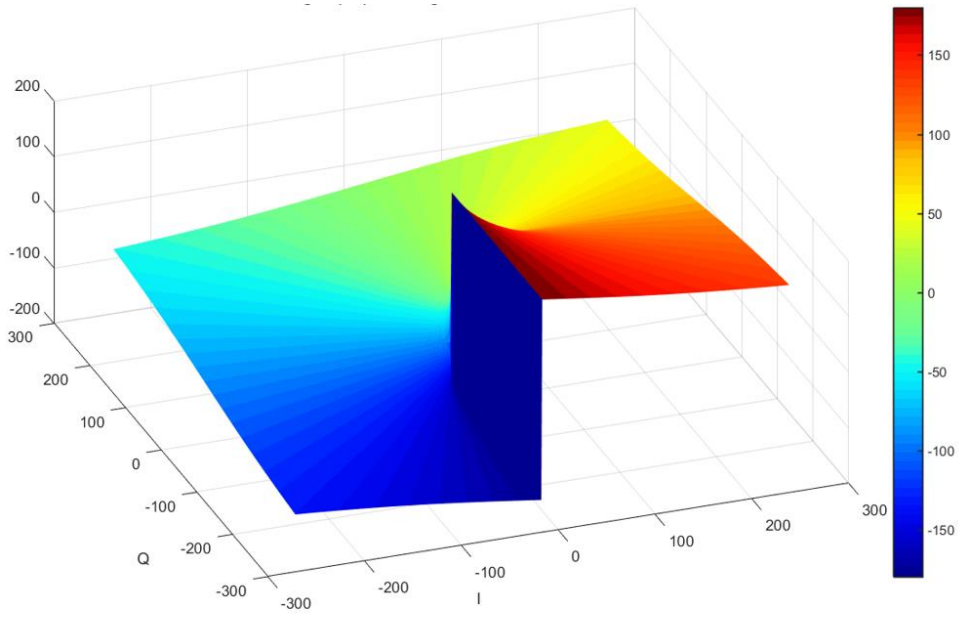


Figure 19. Surf Plot for Arctangent(Q/I) Using Floating-Point Precision Calculation

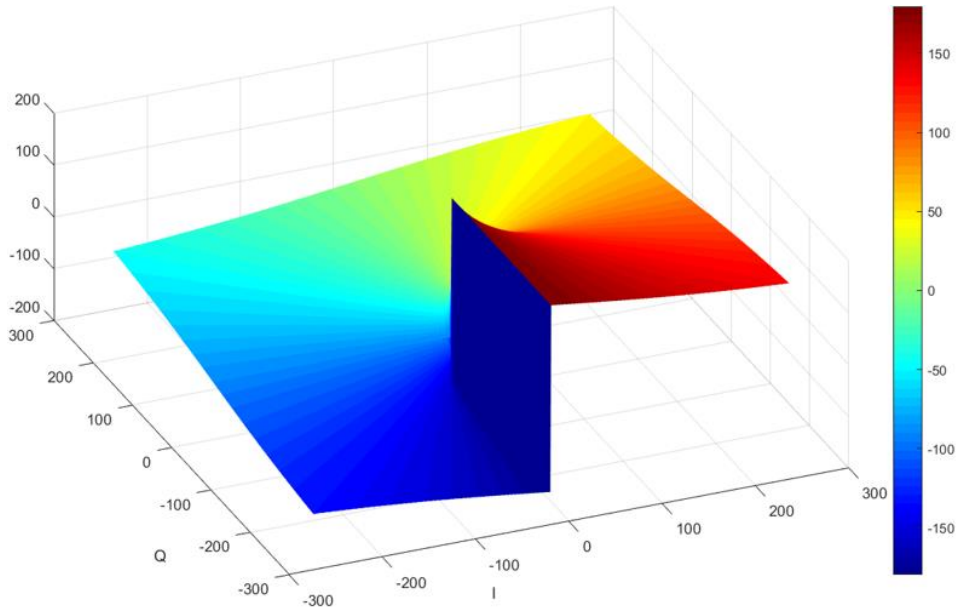


Figure 20. Surf Plot for Arctangent(Q/I) Solved Using Eight-Iteration CORDIC

2. Error Analysis

Taking a closer look at Table 6, we see that the absolute phase error did not always decrease after every rotation. A similar observation is also mentioned in [14]. Nonetheless, there is a positive correlation between the total number of iterations and the accuracy of the CORDIC solution. The angle accumulated by z_i after every rotation is represented by the basic angles that are stored in "*angleLUT*." As these basic angles become smaller and more precise, an increase in the number of iterations allows the subsequent calculation to become more precise as well.

An analysis of this relationship was carried out. The CORDIC algorithm was used to compute the phase angles with I and Q taking on eight-bit binary numbers from all four quadrants. The experiment was repeated with the number of CORDIC iterations varying from one to 18. The maximum phase error was logged for each CORDIC iteration and is presented in Table 8. The results affirm the correlation between iteration and accuracy. In addition, we observe that the magnitude of the error after *i* iterations is less than 2^{-i}, which means the phase angle calculated is accurate to *i* binary digits.

D. FIXED-POINT MODEL

Having verified that the CORDIC algorithm was implemented correctly, we next convert the model to calculate the results of x_i, y_i, and z_i using a fixed number of integer bits and fraction bits. The word format of the CORDIC inputs and output are aligned to the word format used by the DRFM memory and the DIS, as shown in Figure 21. The I and Q inputs are represented using eight integer bits and one sign bit, and the phase output of the CORDIC uses five integer bits to represent 0^o to 360^o.

1. Word Length

The DIS uses the quantization scheme in Table 3 to represent the values of the phase angle. On the other hand, the CORDIC uses five-bit signed numbers to represent angles, which are between -180^o and 180^o. This means if the vector phase angle is 270^o, the CORDIC algorithm returns -90^o. This does not pose a concern to the DIS because the binary expression of 270^o using unsigned five-bit numbers and the binary expression

of -90^{o} using signed five-bit numbers are both '11000'. The quantization scheme of the CORDIC algorithm is listed in Table 9.

Table 8. Maximum Angular Error with Every Incremental Iteration

Iteration	Max absolute phase error (degree)	Max absolute phase error (radian)	Log2(error)
1	45	0.78539816	−0.348503871
2	26.56505118	0.46364761	−1.10889938
3	14.03624347	0.24497866	−2.029271994
4	7.125016349	0.12435499	−3.007463642
5	3.576334375	0.06241881	−4.001875337
6	1.789399349	0.03123091	−5.00088157
7	0.895169329	0.01562365	−6.000124455
8	0.447593141	0.00781197	−7.000097135
9	0.223808221	0.00390619	−8.000022033
10	0.111902544	0.00195307	−9.000042231
11	0.055949652	0.00097651	−10.00008401
12	0.027973199	0.00048822	−11.0001679
13	0.013986868	0.00024412	−12.00014022
14	0.006993631	0.00012206	−13.00009947
15	0.003497044	6.1035×10^{-5}	−14.00000537
16	0.001748515	3.0517×10^{-5}	−15.00001074
17	0.000874253	1.5259×10^{-5}	−16.00001886
18	0.000437121	7.6292×10^{-6}	−17.00003773

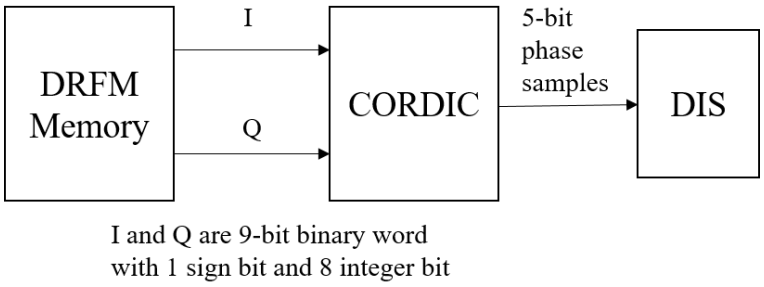


Figure 21. I/O Bit Formats for CORDIC Phase Converter

Table 9. Quantization Scheme for DIS versus CORDIC

DIS Quantization Scheme			CORDIC Quantization Scheme		
Phase Angle (degree)	**Binary Representation**	**Values of Binary Representation**	**Phase output from CORDIC (degree)**	**2s Complement Representation**	**Values of Binary Representation**
0	00000	0	0	00000	0
11.25	00001	1	11.25	00001	1
22.5	00010	2	22.5	00010	2
33.75	00011	3	33.75	00011	3
45	00100	4	45	00100	4
56.25	00101	5	56.25	00101	5
67.5	00110	6	67.5	00110	6
78.75	00111	7	78.75	00111	7
90	01000	8	90	01000	8
101.25	01001	9	101.25	01001	9
112.5	01010	10	112.5	01010	10
123.75	01011	11	123.75	01011	11
135	01100	12	135	01100	12
146.25	01101	13	146.25	01101	13
157.5	01110	14	157.5	01110	14
168.75	01111	15	168.75	01111	15
180	10000	16	−180	10000	−16
191.25	10001	17	−168.75	10001	−15
202.5	10010	18	−157.5	10010	−14
213.75	10011	19	−146.25	10011	−13
225	10100	20	−135	10100	−12
236.25	10101	21	−123.75	10101	−11
247.5	10110	22	−112.5	10110	−10
258.75	10111	23	−101.25	10111	−9
270	11000	24	−90	11000	−8
281.25	11001	25	−78.75	11001	−7
292.5	11010	26	−67.5	11010	−6
303.75	11011	27	−56.25	11011	−5
315	11100	28	−45	11100	−4
326.25	11101	29	−33.75	11101	−3
337.5	11110	30	−22.5	11110	−2
348.75	11111	31	−11.25	11111	−1

The number of integer bits and fraction bits for x_i, y_i, and z_i are determined next. As the iteration carries on, x_i grows and its magnitude converges toward a value that is the magnitude of the input vector times the scale factor of 1.647. The largest magnitude, when I and Q have a magnitude of 255, is

$$|Z|_{\max} = \sqrt{255^2 + 255^2} = (1.414)(255) = (1.414)|\mathrm{I}| \tag{4.20}$$

where $|\mathrm{I}|$ is the magnitude of I. This leaves x_n to be

$$x_n = (1.647)(1.414)(255) = (2.331)|\mathrm{I}|. \tag{4.21}$$

As a result, x_i has to be given two more integer bits, making a total of ten integer bits plus one sign bit in order to have enough accuracy. With every rotation, y_i becomes smaller and requires more precision, and, hence, more fraction bits; otherwise, the next direction of rotation could be wrong and result in an incorrect add/subtract operation of the rotation angle from "*angleLUT*." In addition, z_i also receives one more integer bit, giving it a total of five integer bits and one sign bit. The additional integer bit is required when calculating a vector that has a phase angle close to 180^o or -180^o. As the initial few rotations can all be made in the same direction, at one stage z_i could become larger than 180^o or less than -180^o, which are not representable by a five-bit signed number; hence, the need to introduce an additional integer bit. As z_i converges more, the vector returns to quadrants two or three, and the subsequent results only require four integer bits for representation.

2. Implementation Using MATLAB

The changes made in order to implement a CORDIC processor using fixed number format are the following:

1. The fixed-point numbers are constructed using the 'fi(*v*,*s*,*w*,*f*)' function in MATLAB where '*v*' is the value of origin word, '*s*' indicates whether the fixed-pointe number is signed or unsigned, '*w*' is the length of the entire fixed-point number, and '*f*' is the number of fraction bits.

2. As the phase angles are quantized at 11.25° per bit, the rotation angles in "angleLUT" are divided by 11.25 before being used in the "fi" function as parameter 'v'.

3. The multiplication by 2^{-i} is carried out by using the bit-shift operation – the 'bitsra(*a*, *k*)" function where '*a*' is right-shifted arithmetically by *k* number of bits.

4. After all the rotations are completed, the results are rounded to the nearest integer bit, and the five least significant integer bits are extracted to form the phase result of the CORDIC processor.

3. Simulation and Results

The same complex vector Z = −45 + 23j is used to test the CORDIC processor functionality in using fixed-point numbers. A visual representation of the vector rotation is presented in Figure 22, and the accumulated angles z_i after every iteration are shown in Figure 23. Just like the results from the floating-point model, the vector was progressively rotated toward the positive x-axis, and calculated angles converged toward the theoretical value. The numerical results from every iteration of the CORDIC are presented in Table 10. Here, the values of y_i and the ones from Table 6 share the same sign, implying that the vector was being rotated in the same direction for each iteration in both simulations.

Table 10. CORDIC Calculation for Z = −45 + 23j
Using Fixed-Point Numbers

Iteration	x_i	y_i	z_i	$z_i \times 11.25^\circ$ (degree)	$z_i - z_{i+1}$	$(z_i - z_{i+1}) \times 11.25^\circ$ (degree)	Absolute phase error (degree)
i=0	23	45	8	90	NA	NA	62.9296875
i=1	68	22	12	135	4	45	17.9296875
i=2	79	-12	14.359375	161.5429688	2.359375	26.54296875	8.61328125
i=3	82	7.75	13.109375	147.4804688	-1.25	-14.0625	5.44921875
i=4	82.96875	-2.5	13.7421875	154.6015625	0.6328125	7.119140625	1.671875
i=5	83.125	2.68359375	13.421875	150.9960938	-0.3203125	-3.603515625	1.93359375
i=6	83.20703125	0.0859375	13.578125	152.7539063	0.15625	1.7578125	0.17578125
i=7	83.20703125	-1.2109375	13.65625	153.6328125	0.078125	0.87890625	0.703125
i=8	83.21875	-0.5625	13.6171875	153.1953125	-0.0390625	-0.439453125	0.265625

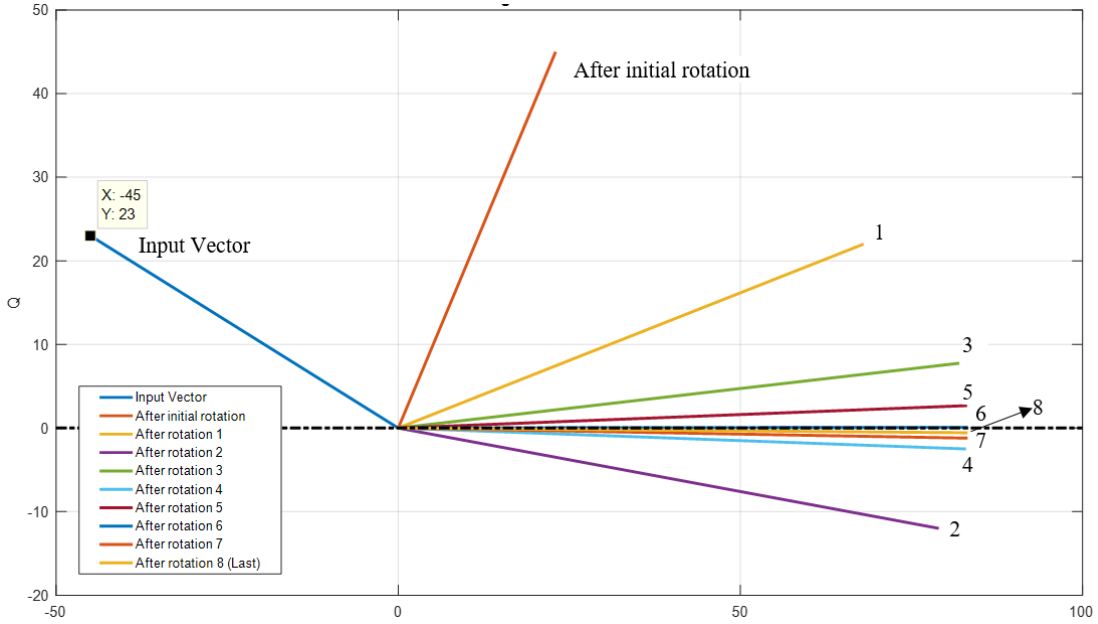


Figure 22. CORDIC Iterations Using Fixed-Point Numbers

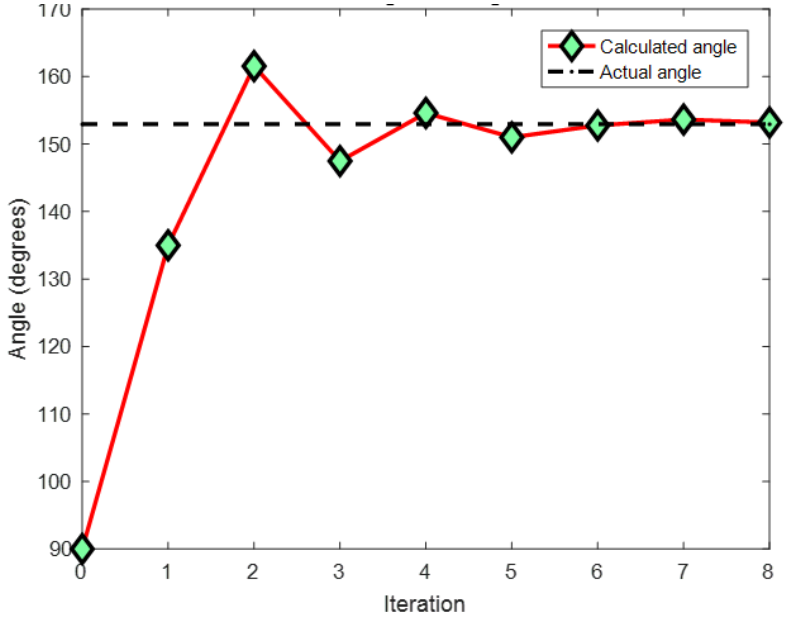


Figure 23. Cumulative Angle through Iterations Calculated Using Fixed-Point Numbers

The binary representation of z_i is shown in Table 11. The phase stored in z_8 was used to derive the final result by following the procedure in Table 12.

Table 11. Binary Representation of z_i for Z = −45 + 23j

Iteration	z_i	Iteration	z_i
i=0	0010000000000	i=5	0011010110110
i=1	0011000000000	i=6	0011011001010
i=2	0011100101110	i=7	0011011010100
i=3	0011010001110	i=8	0011011001111
i=4	0011011011111		

Table 12. Steps to Represent CORDIC Result for Z = −45 + 23j Using Five Bits

S/N	Steps	Result
1	Extract the five least significant integer bits and the most significant fraction bit.	011011
2	If the fraction bit is '1', increase the five-bit number by 1.	01110
3	Multiply decimal representation of the result by 11.25^o to get the phase angle.	$14 \times 11.25^o = 157.50^o$
4	Find the phase error by applying $\tan^{-1}(Q/I) -$ CORDIC result.	$152.927919^o - 157.50^o = -4.572080^o$

To illustrate that the model works for a vector that is in quadrant three, another vector is used: Z2 = −45 − 23j. The binary representation for z_i is shown in Table 13.

Table 13. Binary Representation of z_i for Z = −45 − 23j

Iteration	z_i
i=0	1110000000000
i=1	1101000000000
i=2	1100011010010
i=3	1100101110010
i=4	1100100100001
i=5	1100101001010
i=6	1100100110110
i=7	1100100101100
i=8	1100100110001

The final calculation, as illustrated in Table 14, yields '10010,' which represents 202.50°.

Table 14. Steps to Represent CORDIC Result for Z = −45 − 23j Using Five Bits

Step	Procedures	Result
1	Extract the five least significant integer bits and the most significant fraction bit.	100100
2	If the fraction bit is '1', increase the five-bit number by 1.	10010
3	Multiply decimal representation of the result by 11.25° to get the phase angle.	$18 \times 11.25^{o} = 202.50^{o}$
4	Find the phase error by applying $\tan^{-1}(Q/I)$ − CORDIC result.	$(-152.927919^{o} + 360^{o}) - 202.50^{o} = 4.572080^{o}$

To test the ability of the CORDIC processor to handle interim angles that are larger than 180°, the vector $Z_3 = -230 + 1j$ was used. The numerical result is shown in Table 15. For iteration i = 1, 2, 3 and 4, the vectors were rotated in the same direction, and this caused z_4 to exceed 180°, creating an overflow into the second most significant bit (MSB) which was added to contain this situation. The subsequent iteration featured rotation in the opposite direction, bringing z_5 back below 180° and toward the correct value eventually. After all the rotations were completed, z_8 was rounded to the nearest integer bit, and the CORDIC processor returned −180° or '10000' as the result. The DIS received the phase result as '10000' as an unsigned number and interpreted it as 180°.

A surf plot is displayed in Figure 24, from which we see that the results of the arctangent function solved using the CORDIC algorithm are quantized into 32 values. Based on the results of the test conducted thus far, we assess that the CORDIC processor using fixed-point numbers was implemented correctly.

Table 15. CORDIC Calculation for Z = −230 + 1j
Using Fixed-Point Numbers

Iteration	x_i	y_i	z_i	z_i in binary	$z_i \times 11.25^\circ$ (deg)	$z_i - z_{i+1}$	$(z_i - z_{i+1}) \times 11.25^\circ$ (deg)	Absolute phase error (deg)
i=0	10	230	8	'0010000000000'	90	NA	NA	87.51171875
i=1	240	220	12	'0011000000000'	135	4	45	42.51171875
i=2	350	100	14.35938	'0011100101110'	161.5429688	2.359375	26.54296875	15.96875
i=3	375	12.5	15.60938	'0011111001110'	175.6054688	1.25	14.0625	1.90625
i=4	376.5625	-34.375	16.24219	'0100000011111'	182.7265625	0.632813	7.119140625	5.21484375
i=5	378.7109	-10.8398	15.92188	'0011111110110'	179.1210938	-0.32031	-3.603515625	1.609375
i=6	379.0508	0.992188	15.76563	'0011111100010'	177.3632813	-0.15625	-1.7578125	0.1484375
i=7	379.0625	-4.92969	15.84375	'0011111101100'	178.2421875	0.078125	0.87890625	0.73046875
i=8	379.1016	-1.96875	15.80469	'0011111100111'	177.8046875	-0.03906	-0.439453125	0.29296875

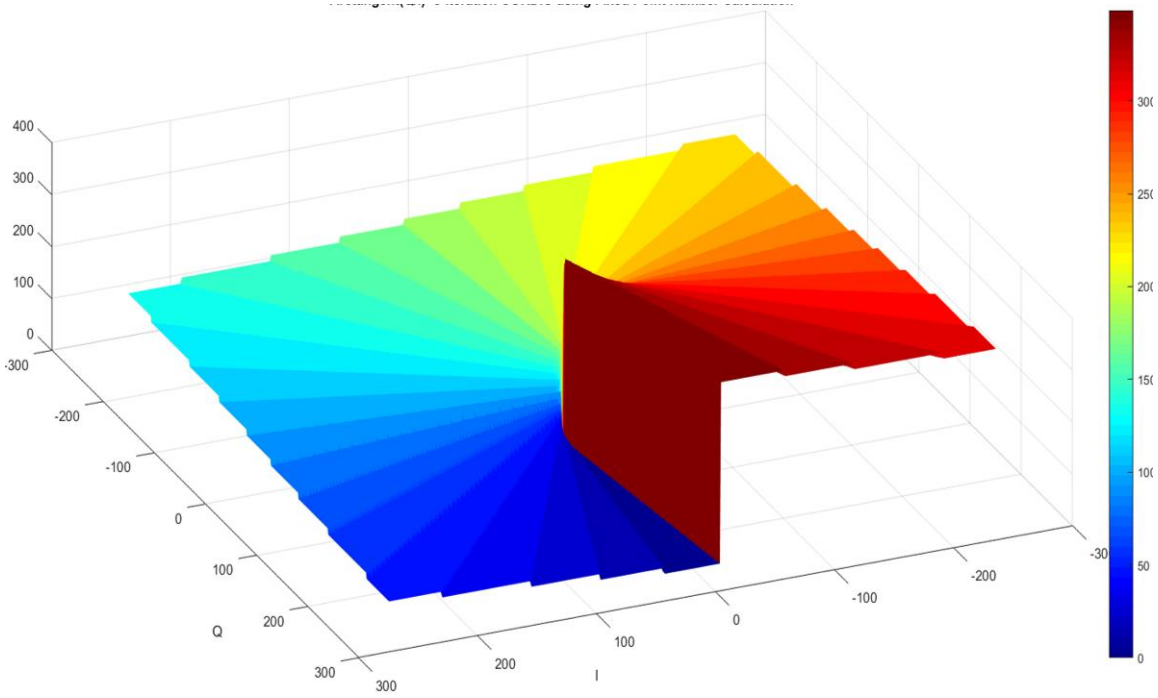


Figure 24. Surf Plot for Arctangent(Q/I) Solved by Eight-Iteration CORDIC
Using Floating-Point Precision Calculation

4. Error Analysis

The requirement on precision is investigated next. Since the last stage of the CORDIC process involves rounding the first fraction bit to the nearest integer bit, the CORDIC processor must be able to resolve 5.625^o. This means that the maximum error must be less than that. The criteria to represent an input vector with an angle around 5.625^o as '00000' or '00001' is illustrated in Figure 25.

The eight-iteration CORDIC algorithm that was used so far is unable to provide that kind of precision. The absolute angular error distributed over the angles between -180^o and 180^o is shown in Figure 26. A magnified version of the figure is shown in Figure 27, in which we see that the CORDIC is not accurate enough in some instances. The accuracy can be increased by increasing the number of iterations and the fraction bits for x_i, y_i, and z_i. The improvement in accuracy when the number of iterations increases from eight to 16 and then to 18 is shown in Figure 28. In fact, after 18 iterations, all the error magnitudes are less than 5.625^o. The maximum errors for five iterations to 18 iterations are presented in Table 16. Based on this result, we decided that the CORDIC processer will use 18 iterations in the design.

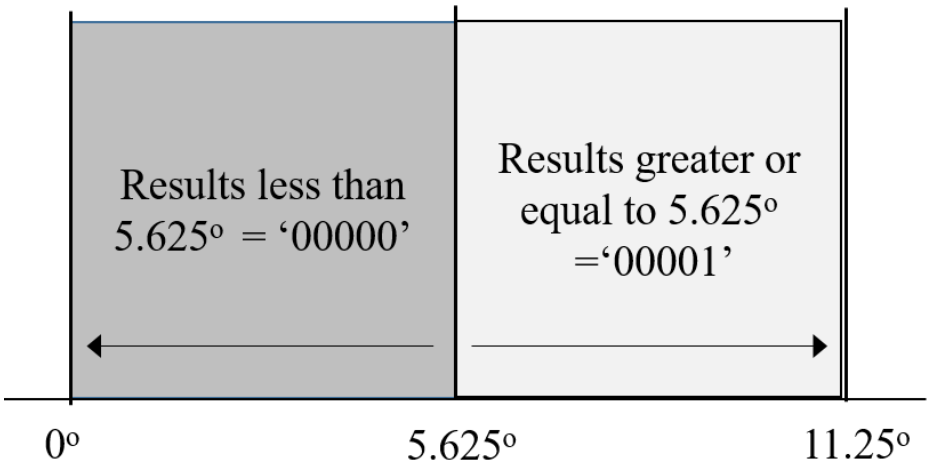


Figure 25. Criteria for Rounding Up or Down

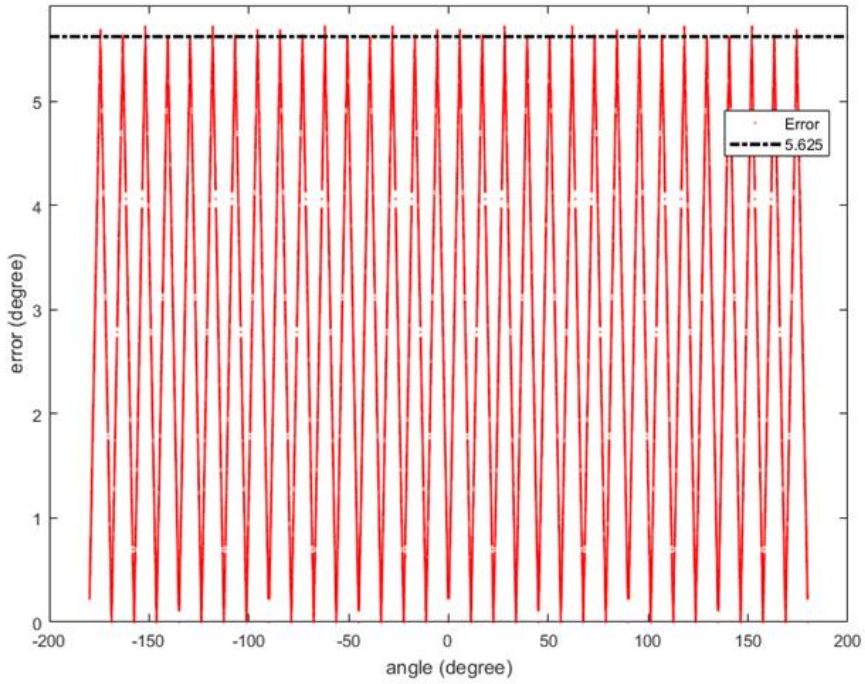


Figure 26. Absolute Angular Error for an Eight-Iteration CORDIC Processer

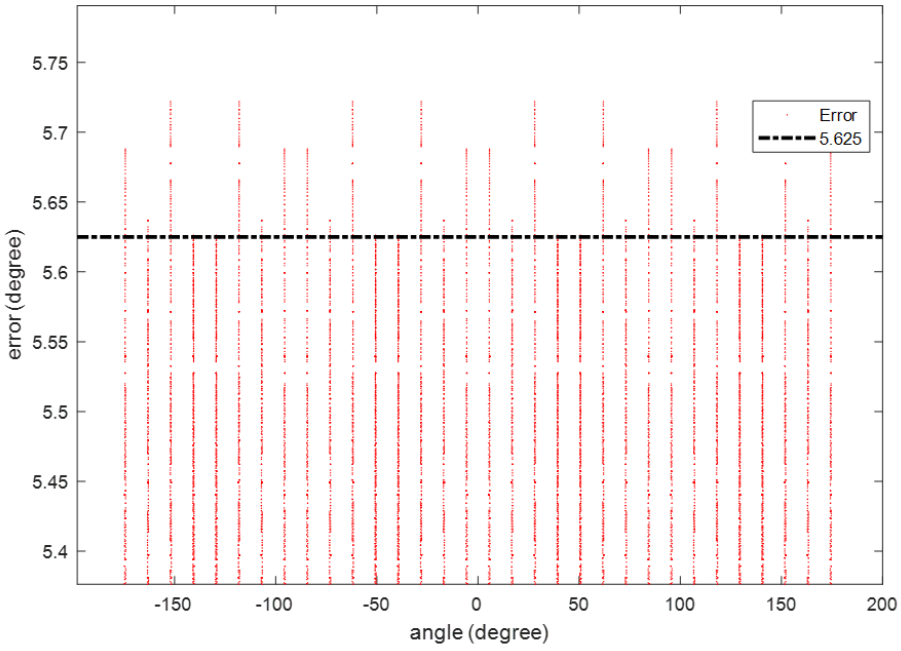


Figure 27. Close-Up View of Figure 25

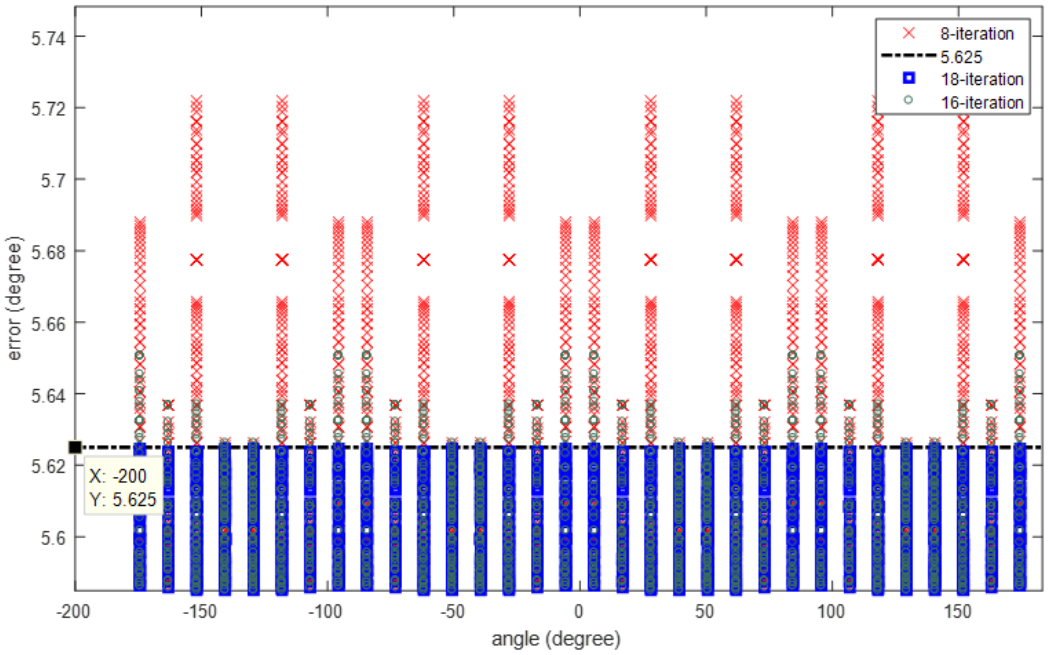


Figure 28. Angular Error for 8-Iteration, 16-Iteration, and 18-Iteration CORDIC Processors

Table 16. Maximum Phase Error for CORDIC Processors with Different Iterations (5 to 18)

Iteration	Max phase error (absolute)
5	8.403824058
6	7.398901839
7	6.421151438
8	5.9876046
9	5.847255441
10	5.721916914
11	5.650660663
12	5.650660663
13	5.636791124
14	5.626312637
15	5.626312637
16	5.626312637
17	5.625371959
18	5.624628041

E. IMPLEMENTATION USING VERILOG

The implementation of the CORDIC algorithm using the Verilog hardware description language is discussed in this section. Verilog was chosen since the DIS has already been implemented in Verilog. Instead of utilizing existing CORDIC Intellectual Property cores created by a Field Programmable Gate Array original equipment manufacturer such as Xilinx or Altera for their own products, the codes that describe the CORDIC processor were crafted based on the CORDIC Equations (4.13), (4.14), and (4.15). The advantage of doing this is that the hardware implementation does not have to be tied to a specific hardware platform.

1. Overview

The Verilog code for CORDIC processing is adapted from an open source version written for implementing the CORDIC algorithm in the rotation mode [15]. The code uses behavioral modeling that shares a similar coding style with MATLAB. In fact, the MATLAB code and the Verilog read very similar to each other and only differ in a few ways. First, the signs of x_i and y_i are determined by their MSB, which is a '0' for a positive 2s complement number and '1' for a negative number. Second, every rotation is triggered by an event, which in this case is the positive edge of the rising clock. Third, registers are used to hold the values for x_i, y_i, and z_i, and to pass them to the next stage of processing, which results in the CORDIC processor having a pipelined architecture. The Verilog code for the CORDIC processor can be found in Appendix B.

2. Simulation and Results

The simulation began after the Verilog codes for the CORDIC processor and its test bench files were completed and able to be compiled. The test cases and results are described in the following subsections.

a. Input: $Z = -45 + 23j$

The same complex vector previously used to test the MATLAB models was used again to test the Verilog model. The CORDIC processor was able to return the phase result as '01110', which is identical to the one generated by the MATLAB model. The

values of x_i, y_i, and z_i from the Verilog model were converted to an unsigned number and compared with those from the MATLAB model. The results, featured in Figure 29, show that these values are identical. The screenshot of the simulation in ModelSim is presented in Figure 30.

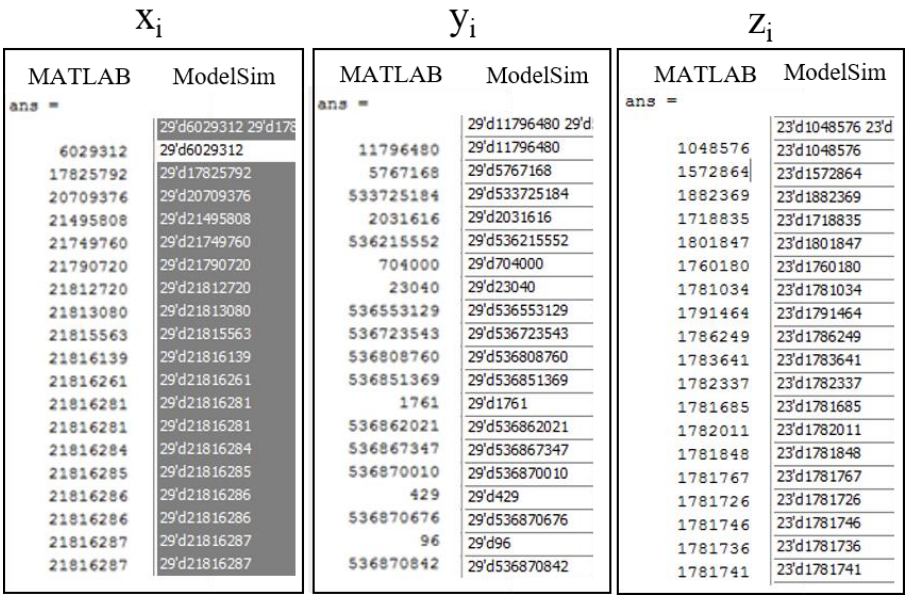


Figure 29. Comparison of x_i, y_i, and z_i Calculations by MATLAB and Verilog

b. Input: I = 0, Q = 0

The next simulation aimed to test the ability of the CORDIC processor to calculate the phase angle when the inputs are the origin coordinates. While the values for x_i and y_i remain at zero, the sign of y_i is decided as '0' during each iteration, causing z_i to grow by accumulating more rotation angles from the "*angleLUT*." This is avoided by forcing the final phase output to be zero if x_{18} is checked and found to be zero. The screenshot of the simulation in ModelSim is presented in Figure 31. Here, we see that the phase output generated is zero after 18 positive-rising edges of the clock.

c. Pipeline

To test this critical feature, three inputs were clocked consecutively over three clock cycles. The screenshot of the simulation in ModelSim is presented in Figure 32,

which shows that 18 clock cycles later, the three results emerged from the CORDIC processor over three consecutive clock cycles in a first-in-first-out fashion.

d. Input: All Possible Combinations

The final test involves using all the possible nine-bit integer values as input. The results are then compared to results generated by MATLAB and show a complete match.

F. CONCLUSION

The design and testing of an I/Q phase converter was discussed in this chapter. The result was an 18-iteration CORDIC processor which takes 18 clock cycles to generate phase samples quantized at five bits for the DIS in a pipelined manner. The simulation of sea clutter to improve the false target image is examined in the next chapter.

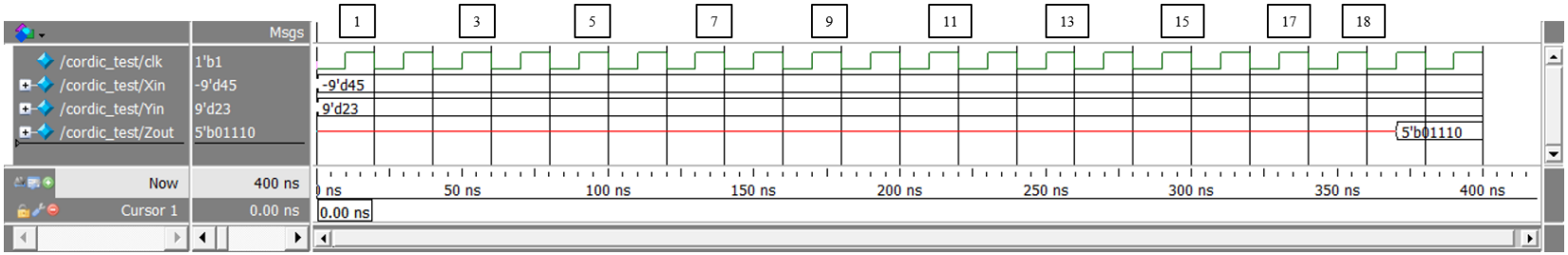


Figure 30. ModelSim Simulation Using I=−45, Q=23 with Phase Result (Zout) Showing 5’b01110

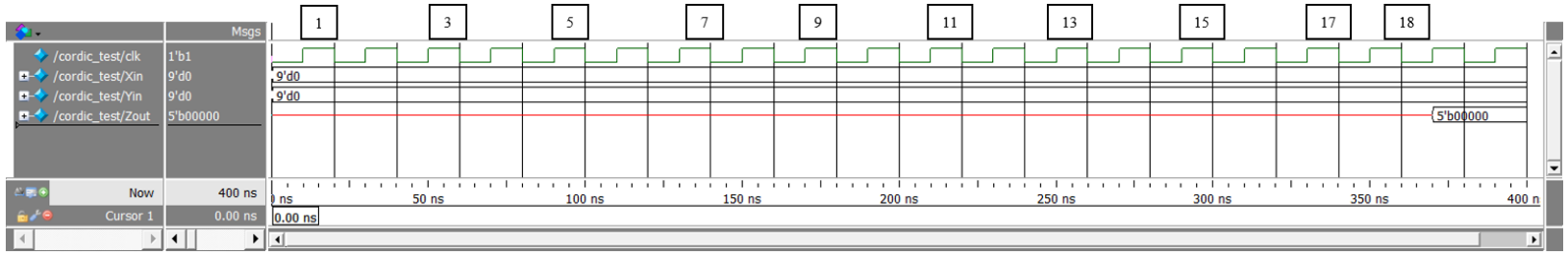


Figure 31. ModelSim Simulation Using I=0, Q=0 with Phase Result (Zout) Showing 5’b00000

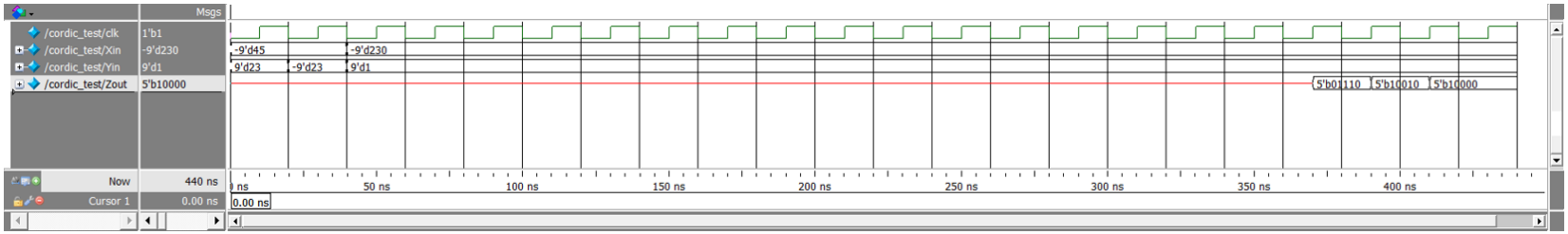


Figure 32. ModelSim Simulation Showing Pipelined Output

V. SEA CLUTTER TARGET PROFILE

The characteristics that an ISAR operator can use to identify a false image synthesized by the DIS are described in Chapter I. The false target profile is built with a limited number of point scatterers and does not include the environment such as the clutter return and the noise return. In this chapter the inclusion of a sea clutter profile in the DIS with the aim of making the false target appear realistic is discussed.

A. COHERENT SEA CLUTTER SIMULATIONS

The sea clutter model used in this thesis is adapted from the research done by previous students and scientists from the Center for Joint Services Electronic Warfare. The sea clutter model was developed initially by Brooks [16], who studied the works of Ward, Watts, and Walker. The model was subsequently improved by Dr. Sebastian Teich, a visiting scientist from Germany, to support the design of an antenna for surface vessels. With help from Dr. Teich, the model was modified to fit the context of a high-resolution radar mounted on an airborne platform. In the model, the amplitude and Doppler of the sea clutter are random variables that fluctuate according to two separate random distributions.

1. Clutter Amplitude Model Using the KA Distribution

In his sea clutter model, Brooks used a KA distribution to calculate the return sea clutter power [16]. The model allows the user to simulate the sea clutter by specifying the sea state condition, RF polarization, a normalized RCS model, and a wind direction. In this research, a horizontal polarization and a normalized RCS model developed by the Naval Research Laboratory [17] is used. Unlike a surface radar, an airborne radar has a different grazing angle as displayed in Figure 33.

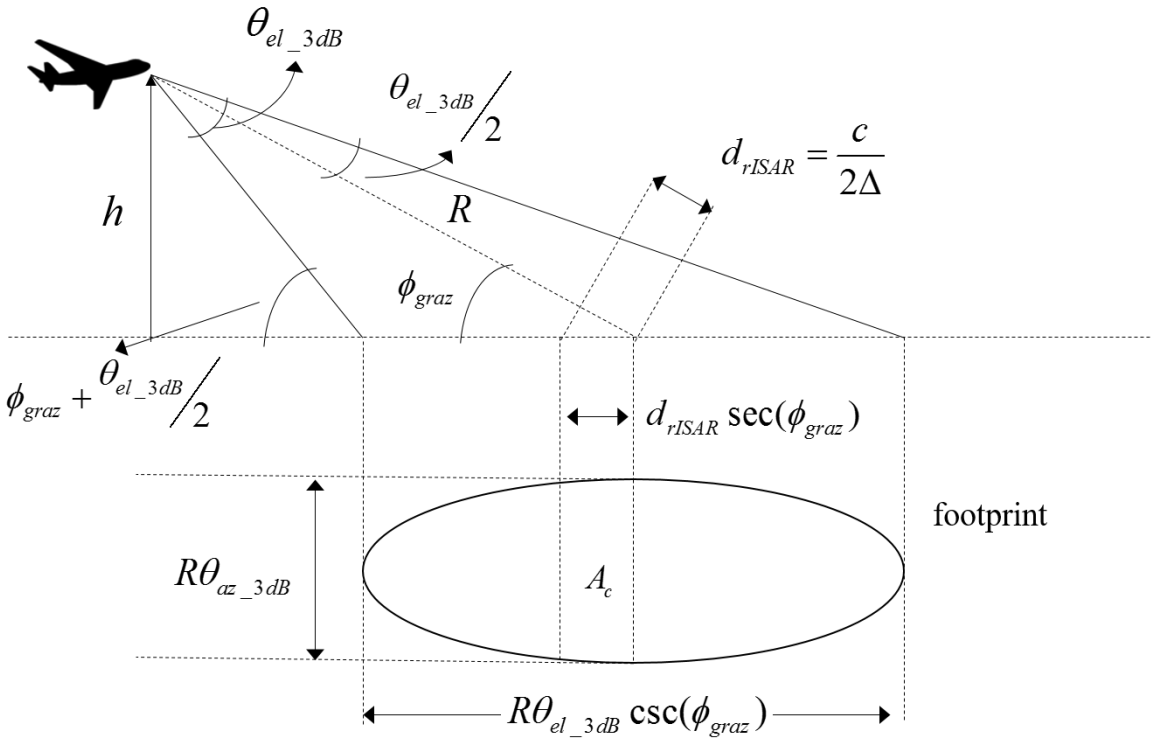


Figure 33. Clutter Geometry for Airborne ISAR. Adapted from [18].

From [20], given an aircraft at height *h* above the ground, the grazing angle ϕ_{graz} at a given range *R* can be expressed as

$$\phi_{graz} = \sin^{-1}\left(\frac{h}{R} + \frac{h^2}{2r_e R} - \frac{R}{2r_e}\right) \tag{6.1}$$

where r_e is the effective Earth radius. The relationship between *R* and ϕ_{graz} at a constant *h* is shown in Figure 34. The area of the illuminated patch A_c can be approximated as

$$A_c = R\theta_{az\_3dB} d_{rISAR}\sec(\phi_{graz}) \tag{6.2}$$

where d_{rISAR} is the slant range resolution and θ_{az_3dB} is the ISAR azimuth beamwidth. This relation is shown in Figure 34.

a. The NRL Normalized RCS Model

The radar range equation is used to calculate the mean clutter return power P_{RC}^{o} and is expressed as

$$P_{RC}^{0} = \frac{P_T G_t G_r \lambda^2 \sigma^0 A_c}{(4\pi)^3 R^4 L} \tag{6.3}$$

where σ^0 is the normalized mean sea clutter scatter coefficient, which can be obtained using the NRL model, and G_t and G_r are the transmit and receive gain, respectively. Also, P_T is the peak power of the pulsed-ISAR waveform transmitted, *L* represents losses, and λ is the carrier wavelength. The NRL model describes an empirical relationship, which is expressed as

$$\sigma_{HH,VV}^{0} = c_1 + c_2 \log_{10} \sin\left(\phi_{graz}\right) + \frac{(2.75 + c_3\phi_{graz}) \log_{10} f_{GHz}}{(1 + 0.95\phi_{graz})} + c_4 (1 + SS)^{\frac{1}{2+0.085\phi_{graz}+0.033SS}} + c_5 \phi_{graz}^2 \tag{6.4}$$

where $\sigma_{HH,VV}^{0}$ is the normalized co-polarized RCS, *SS* is the Douglas sea state, f_{GHz} is the carrier frequency in GHz, and c_1, c_2, c_3, c_4, and c_5 are free parameters that are adjusted to minimize the average absolute deviation between the empirical equation and a set of data collected by Nathanson [17]. The Douglas sea state describes the roughness of the sea wave surfaces and is shown in Table 17. The free parameters values from the NRL model are shown in Table 18.

Table 17. Summary of Sea State. Source: [17].

Sea State	Description	Wave height (ft)	Wind speed (kn)	Fetch (nmi)	Duration (h)
1	Smooth	0-1	0-6		
2	Slight	1-3	6-12	50	5
3	Moderate	3-5	12-15	120	20
4	Rough	5-8	15-20	150	23
5	Very rough	8-12	20-25	200	25
6	High	12-20	25-30	300	27
7	Very High	20-40	30-50	500	30

Table 18. Summary of NRL Model Free Parameters. Source: [17].

	Polarization	
Constants	Horizontal	Vertical
c_1	− 73.00	− 50.79
c_2	20.78	25.93
c_3	7.351	0.7093
c_4	25.65	21.58
c_5	0.00540	0.00211

b. Clutter Parameter Comparison

The inputs for the NRL RCS model are grazing angle, RF polarization, carrier frequency, and sea state condition. For this thesis research, the RF polarization and the carrier frequency are fixed to horizontal polarization and 10 GHz, which are normally the specifications for an ISAR radar such as the APS/137 ISAR. The effect of the grazing angle and sea state on the normalized RCS are presented in this section.

(1) Grazing Angle

Assuming that the height of the aircraft *h* remains constant, we see that the grazing angle ϕ_{graz} decreases as the slant range *R* increases. This is illustrated in Figure 34 with the main objective to show the relationship between *R* and ϕ_{graz}. The effect this has on σ^0 is illustrated in Figure 35 where σ^0 decreases as ϕ_{graz} increases. The RCS of a clutter patch σ_c is the product of A_c and σ^0, or $\sigma_c = A_c\sigma^0$. The relationship between σ_c and *R*, assuming *h* is constant, is displayed in Figure 36. The illuminated area A_c increases significantly less and results in an overall decrease in the clutter patch σ_c.

(2) Sea State

The effect the sea state *SS* has on σ^0 is displayed in Figure 37, which shows the normalized RCS as a function of *R* for sea state 2 through 7. Here, it can be seen that an increasing *SS* leads to an increase in σ^0 as well. Based on (6.3) the increase in σ^0 consequently increases the P_{RC}^o as well and is shown in Figure 38 as a function of range *R*.

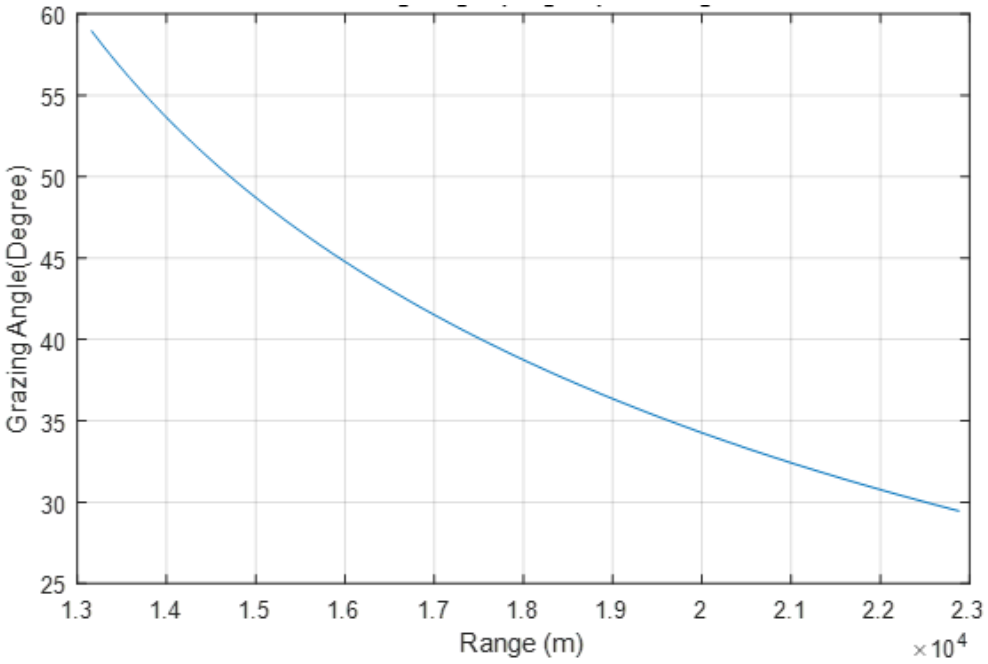


Figure 34. Grazing Angle versus Range

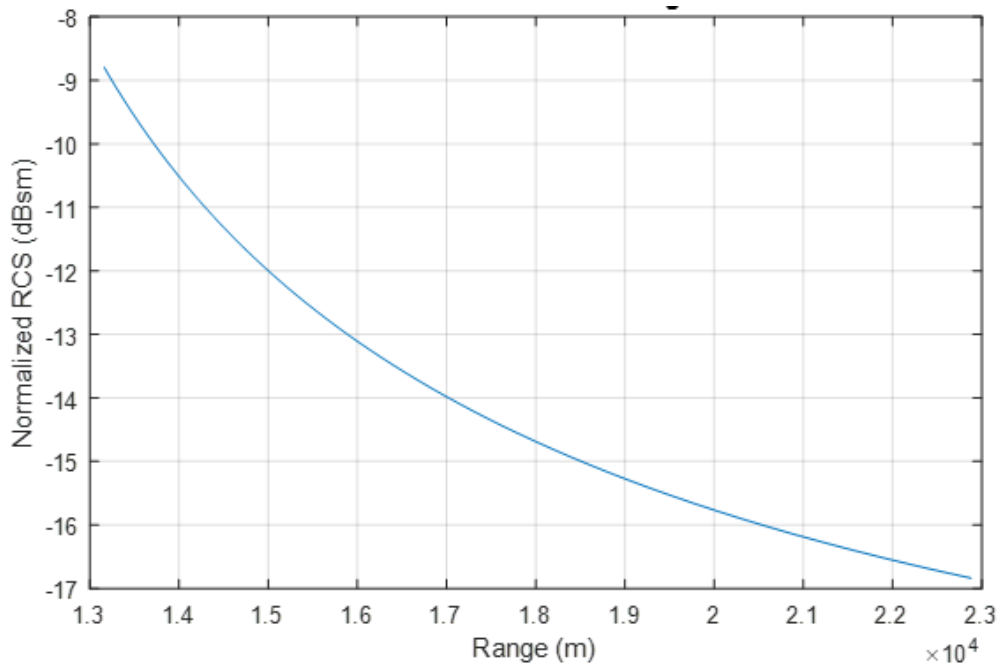


Figure 35. Normalized RCS versus Range

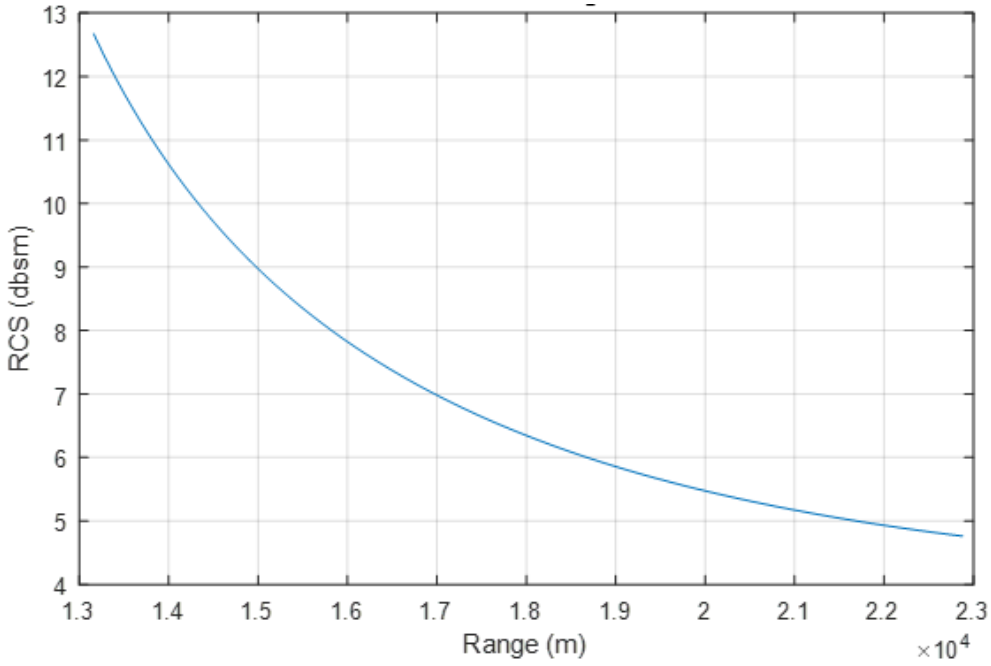


Figure 36. RCS versus Range

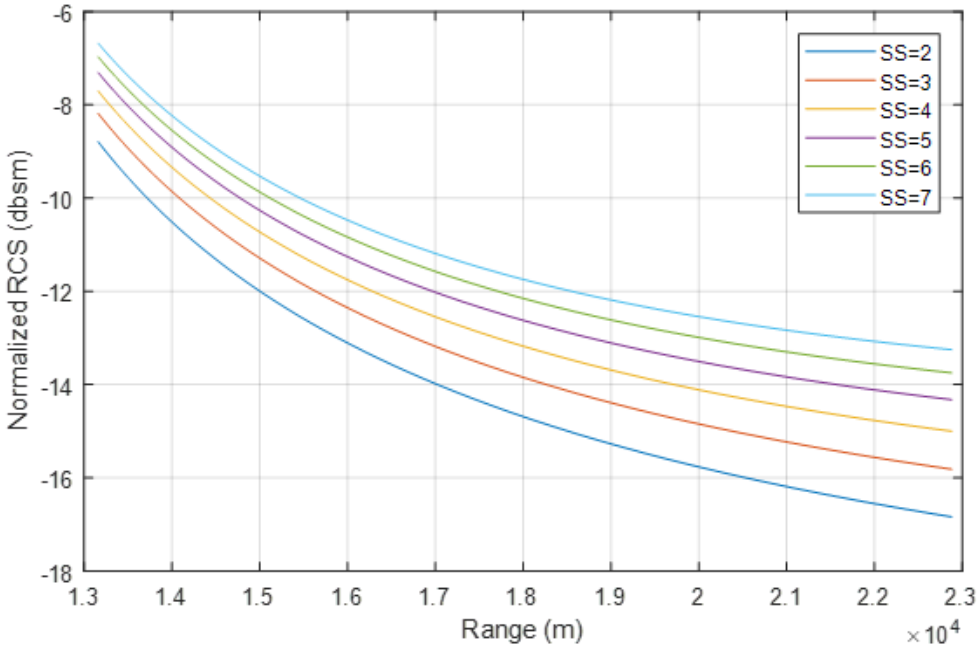


Figure 37. Normalized RCS for Sea Clutter at Different Grazing Angles and SS

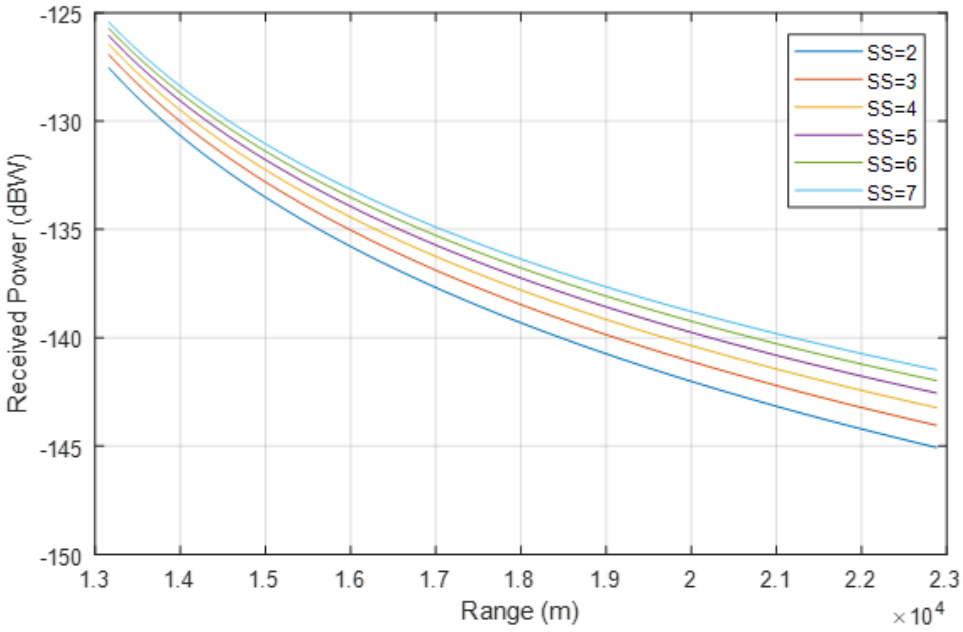


Figure 38. Received Power from Sea Clutter at Different SS

c. Power fluctuation via Compound KA Distribution

The sea clutter simulation model developed by Brooks is comprised of a slow moving component called texture that represents the local sea structures such as Bragg scatterers and distributed white caps [16], [19]. It also includes a fast moving component called speckle that represents structures such as sea spray, discrete white caps, and bursts [20]. The texture is modeled using a gamma distribution, and the speckle follows an exponential distribution. The final expression for the fluctuating sea clutter power is expressed as

$$P(z \mid x) = \left(\sum_{n=0}^{\infty} \frac{1}{x(1+n\rho)} \exp\left(\frac{-z}{x(1+n\rho)} \right) \right) P_{poiss}(n) \tag{6.5}$$

where x is the texture, z is the clutter power, n is the number of instantaneous bursts found in a range cell [19], [21], and ρ is the ratio of the burst power to the Bragg power. A Poisson distribution is used to describe the occurrence of n, which is approximated by

$$P_{poiss}(n) = \begin{Bmatrix} 1-\bar{N}, n=0 \\ \bar{N}, \quad n=1 \\ 0, \quad n \geq 2 \end{Bmatrix} \tag{6.6}$$

where $\bar{N}$ is the probability of a spike.

2. Modeling of Fluctuating Sea Clutter Spectra

The model for the Doppler spectrum for a single range cell is [16]

$$G(v,x,s)=\frac{x}{\sqrt{2\pi s}}\exp\left(\frac{-(v-m_f(x))^2}{2s^2}\right) \tag{6.7}$$

where *v* is the Doppler frequency, $m_f(x)$ is the mean Doppler frequency as a function of texture *x*, and *s* is the standard deviation. The mean frequency can be expressed as

$$m_f(x)=\left(\alpha+\beta\frac{x}{P_{RC}^0}\right)\cos\theta_w+f_D \tag{6.8}$$

where α and β are expressed as [16], [22], [23]

$$11\beta=\alpha+\beta=\frac{2}{\lambda}\left(0.25+kU\right) \tag{6.9}$$

where $k=0.25$ for horizontal polarization and 0.18 for vertical polarization [16], [22]. The Doppler shift introduced by the motion of the transmitting platform is f_D and is set to zero for the relative motion to the clutter as is the case for a stationary ISAR radar. The wind velocity *U* is approximated as [16]

$$U=3.16\left(SS^{0.8}\right). \tag{6.10}$$

The standard deviation of the clutter spectrum *s* follows a Gaussian distribution as

$$p(s)=\frac{\alpha}{\sqrt{2\pi s}}\exp\left(\frac{-(s-m_s)^2}{2\sigma_s^2}\right) \tag{6.11}$$

where $\sigma_s\approx 20$ Hz, and the mean m_s is given by

$$m_s=0.2\frac{U\cos\theta_w}{\lambda} \tag{6.12}$$

where θ_w is the headwind direction.

3. Generating Random Sea Clutter Power and Doppler Spectrum

The fluctuating sea clutter power spectral density is generated when the mean clutter power in (6.3) is applied to the distribution described in (6.5) and (6.7). The mean

clutter power and the fluctuating power for each range bin within the radar main beam is shown in Figure 39. The power spectral density, an example of which can be seen in Figure 40, provides the information on phase and amplitude which can be used to create the phase coefficient and gain coefficient for the sea clutter profile to improve the DIS. The MATLAB code created to simulate the sea clutter for the DIS is found in Appendix A and requires MATLAB code, KdistributionTexture.m, in order to work.

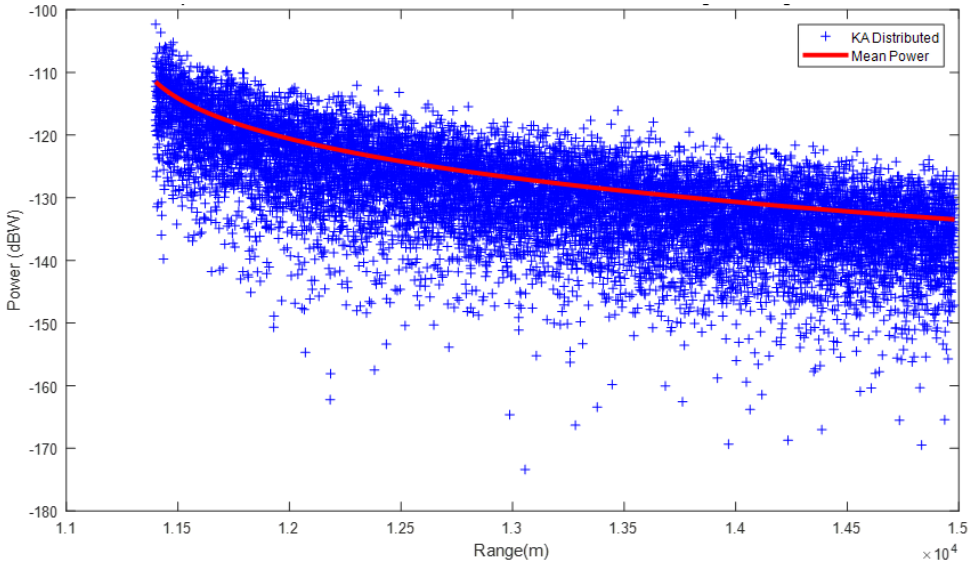


Figure 39. Mean Power and Fluctuating Power of Sea Clutter

B. PHASE AND GAIN COEFFICIENTS FOR THE SEA CLUTTER

In this section the simulation of sea clutter returns to an ISAR radar and the extraction of the phase and the amplitude information from the sea clutter spectrum to create the phase and gain coefficients are discussed.

In Table 19 contains a set of radar parameters that are used to simulate the sea clutter return is given. Information on the operating environment is also included in the table. The radar parameters used are similar to that of the APS 137 ISAR radar [4].

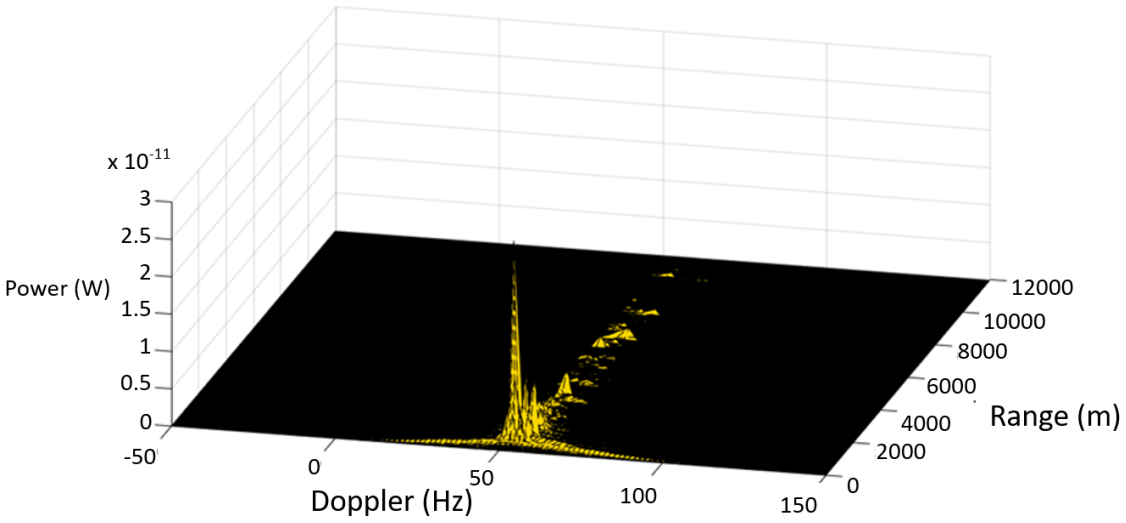


Figure 40. Power Spectral Density of Sea Clutter at *SS=2*

Table 19. Radar Parameters and Operating Environment

Parameter	Values	Parameter	Values
Frequency	10 GHz	Elevation Beamwidth	4.5°
Range Resolution	0.3 m	Azimuth Beamwidth	1.05°
Transmit Gain	32 dB	PRF	200 Hz
Receive Gain	32 dB	Bandwidth	500 MHz
Loss	3 dB	Pulse integration	128
Power	500 W	Polarization	Horizontal
Operating Environment			
Range	3,000 m	Radar height	8,000 ft
Grazing Angle	54.4°	Head wind direction	20°
Sea state	2		

The sea clutter power density spectrum attributed to the radar main beam is shown in Figure 41. The target, which spans 32 range bins, is assumed to be located in the center of the radar beam. The spectra at these range bins are extracted and enlarged in Figure 42.

The power spectral density diagrams for range bins 7, 13, 22 and 32 are shown in Figure 43. The spectra appear different in shape, with range bins 13, 22, and 32 showing a Gaussian outline while range bin 7 resembles a discrete target. They also have different

degrees of ‘spikiness.’ Such wide-ranging differences reflect the probabilistic nature of the complex wave surfaces of the sea clutter model.

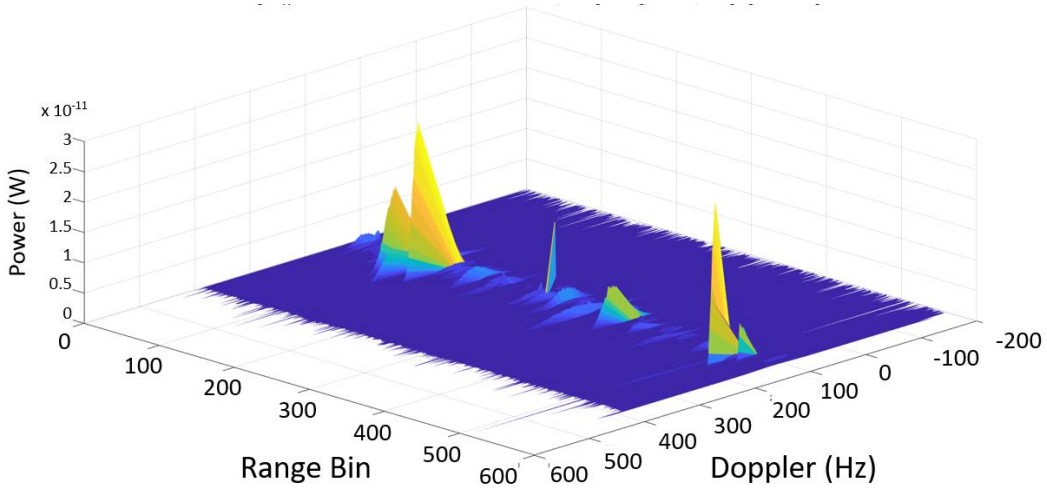


Figure 41. Power Density Spectrum of Sea Clutter

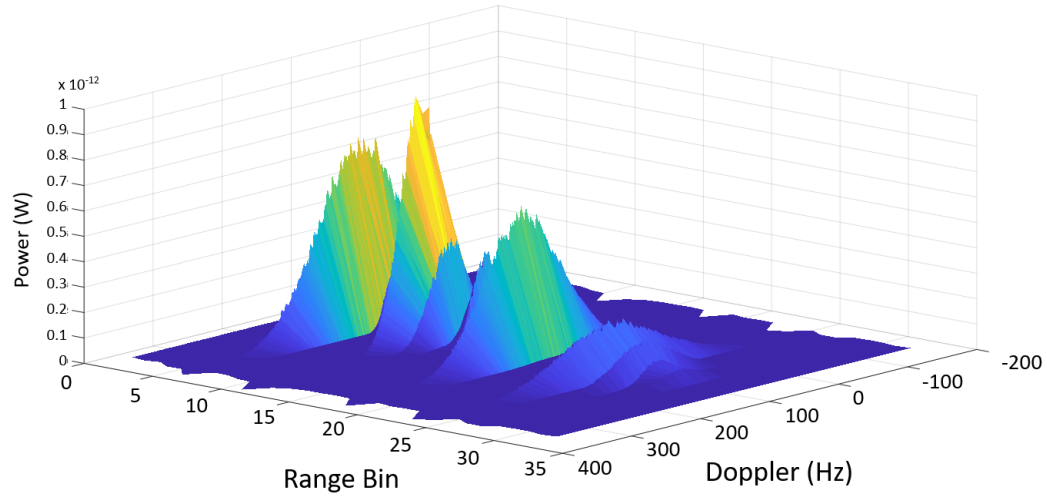


Figure 42. Power Density Spectrum of Sea Clutter in Range Bins Where Target Resides

The power and Doppler components of the spectrum for each range bin are saved and exported to extract_para_Ship_v5.m where they are used to reconstruct the sea clutter return. Just like the target return in (3.1), the clutter return is in complex form and can be represented as a function of the transmitted pulse index and coordinate location on the range-Doppler profile map. It can also be consolidated at each range bin for each pulse by

$$ClutterSum = \sum_{d=1}^{N_d} Clutter(r,d,n). \tag{6.13}$$

The clutter return is added to the target return to form a new complex signal $T''(r,n)$, which is then used to generate the phase coefficient and gain coefficient. The new complex signal is expressed as

$$T''(r,n) = T'(r,n) + ClutterSum(r,n). \tag{6.14}$$

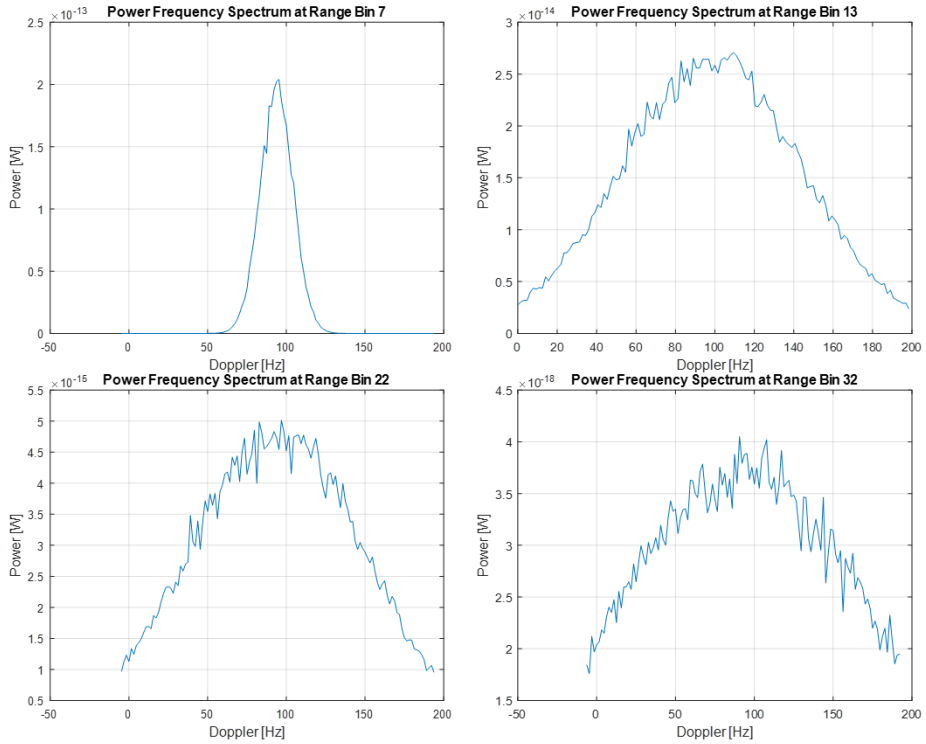


Figure 43. Power Spectrum for Range Bins 7, 13, 22, 32

1. Generating the Phase Coefficient with Sea Clutter

The phase coefficient for $T''(r,n)$ is generated in the same way as is the phase coefficient for $T'(r,n)$ described in Chapter III. The phase angle of $T''(r,n)$ is derived by using its real and imaginary components

$$\phi'_T(r,n) = \measuredangle\left(\frac{imag\{T''(r,n)\}}{real\{T''(r,n)\}}\right) = \measuredangle\left(\frac{imag\{T'(r,n)+ClutterSum(r,n)\}}{real\{T'(r,n)+ClutterSum(r,n)\}}\right). \quad (6.15)$$

Subsequently, the phase coefficient can be generated before it is quantized at five-bits

$$\phi''_{inc}(r,n) = \phi''_{inc}(r,n-1) + \phi'_T(r,n-1) - \phi'_T(r,n). \quad (6.16)$$

2. Generating the Gain Coefficient with Sea Clutter

Similarly, the gain coefficient is generated using the same method as described in Chapter III as

$$T''_N(r,n) = \frac{|T''(r,n)|}{\max\left(|T''(r,n)|\right)} = \frac{|T'(r,n)+ClutterSum(r,n)|}{\max\left(|T'(r,n)+ClutterSum(r,n)|\right)}. \quad (6.17)$$

3. Simulation and Results

The phase and gain coefficients derived from (6.16) and (6.17) are used to modulate the phase samples of the ISAR LFM signal. The output from the DIS is returned to the DRFM and subsequently transmitted back to the ISAR. The resultant range-Doppler images are discussed in the following subsection.

(1) Combined Image

The ISAR image that describes the range-Doppler profile of the target plus the sea clutter $T''(r,n)$ is shown in Figure 44. Initially, the amplitude of the ship $T'(r,n)$ is much larger compared to that of *ClutterSum(r,n),* causing the sea clutter to be insignificant. This is caused by the estimates of the target ship RCS being too large. A more accurate set of RCS values is found from a high-resolution range profile generated using CST Microwave, an EM simulation software tool that uses a three-dimensional Computer Aided Design model of a transport ship. This is related once again to the

modeling of the false target and is reserved for future study. The inclusion of the sea clutter has blurred the edge of the ship to some extent. The range-Doppler image of the sea clutter on its own is shown in Figure 45. Unlike the ship target, the sea clutter remains continuous in the Doppler axis.

(2) Effect of Different Sea States

The effect that sea state has on the sea clutter image is investigated next. The sea clutter at sea states 3 and 4 was extracted and separately combined with the target return of the ship to form two more range-Doppler images. The images for the ship target without sea clutter and with sea clutter at sea states 2–4 are consolidated in Figure 46 for comparison. We see that at higher sea states, more streaks of sea clutter appear on the image, which can be explained by the higher level of power return due to the increase in σ^0. In higher sea states, the velocity of the sea clutter and target increases. For this simulation, the target is fixed in Doppler.

(3) Doppler Resolution

The sea clutter shown in Figure 42 is created based on a Doppler resolution $\Delta f_{clutter}$ of 1.5625 Hz and is able to appear continuous to an ISAR sharing the same Doppler resolution Δf, as seen in Figure 45. If the ISAR increases its Δf further, the sea clutter starts to display discontinuity as can be seen in Figure 47. The continuity in the Doppler axis is restored by remodeling the sea clutter based on a higher Doppler frequency resolution. As shown in Figure 48, when $\Delta f = 0.78125$ Hz, the sea clutter remodeled with $\Delta f_{clutter} = 0.78125$ Hz remained continuous, while the banding gaps widened for the ship target.

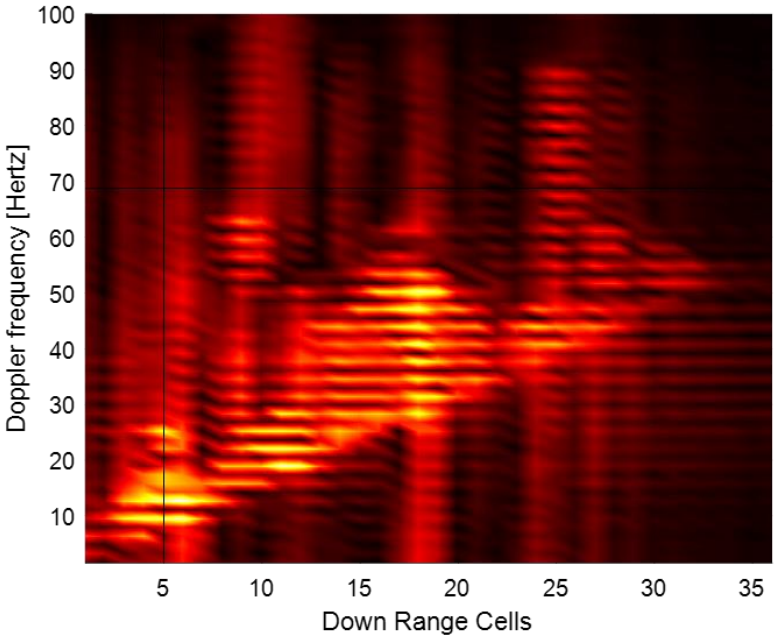


Figure 44. Range-Doppler Test Image of False Target and Sea Clutter

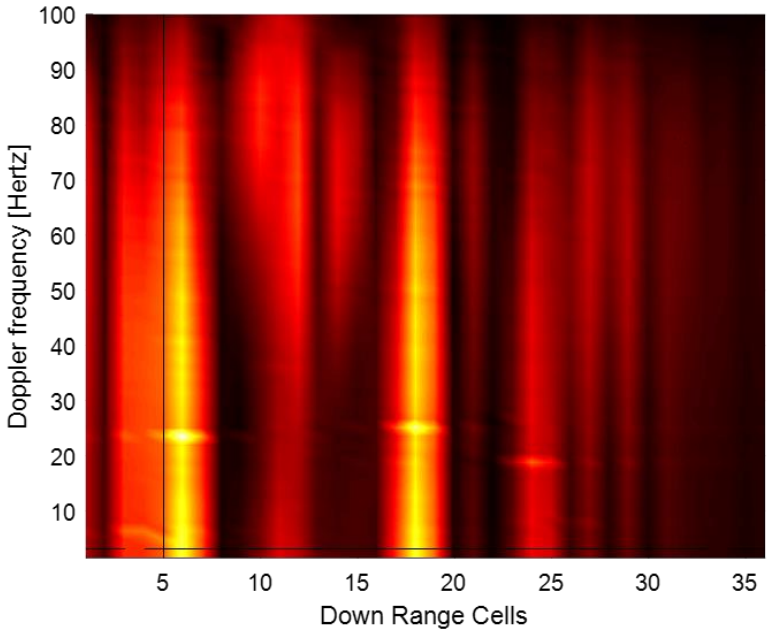


Figure 45. Range-Doppler Test Image of Sea Clutter

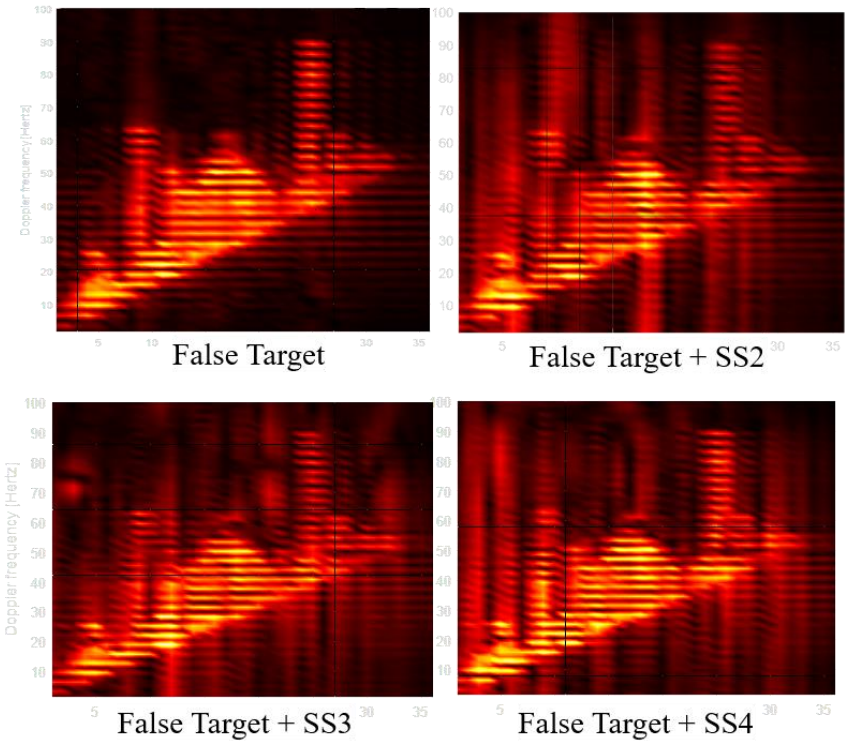


Figure 46. Range-Doppler Test Images at Different Sea States

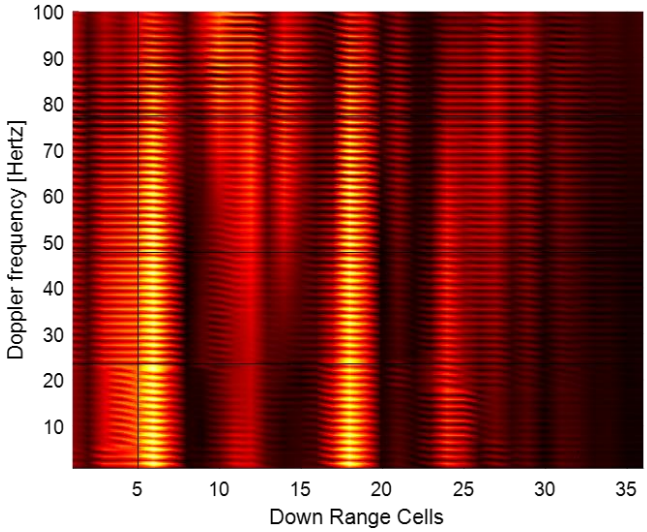


Figure 47. Sea Clutter ($\Delta f_{clutter} = 1.5625$ Hz) Showing Discontinuity When $\Delta f = 0.78125$ Hz

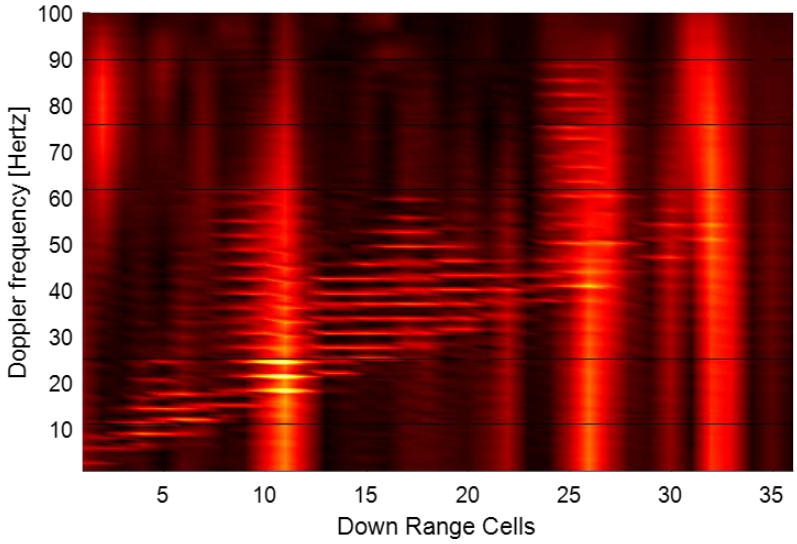


Figure 48. Sea Clutter ($\Delta f_{clutter} = 0.78125$ Hz) Showing Continuity When $\Delta f = 0.78125$ Hz

(4) Number of Doppler Samples

The Doppler components at each individual range bin contribute to the formation of the range-Doppler profile; therefore, the number of samples taken from the power spectrum density determines the number of Doppler components and the appearance of the sea clutter on the range-Doppler image which can be seen in Figure 49. In general, the fewer the samples, the shorter the sea clutter appears.

(5) Creating a Blurry Ship Target

If the desired intent is just to blur the edge of the ship target without oversaturating the entire ISAR image, this can be achieved by using only the sea clutter return, which has similar Doppler frequencies to those of the ship target, to create $T''(r,n)$. As an example, suppose the minimum and maximum Doppler component of the ship target in each range bin is $\min\{d_i^T\}$ and $\max\{d_i^T\}$, respectively, where d_i^T is the set of all the Doppler frequencies that are present in a range bin *i*. The sea clutter returns with Doppler frequencies that are greater than $\min\{d_i^T\} - \Delta f$ and smaller than

$\max\{d_i^T\} + \Delta f$ can be used to construct $T''(r,n)$, while the Doppler frequencies falling outside this range are discarded. The result can be seen in Figure 50, in which the sea clutter has managed to make the edge of the ship target appear blurred. In fact, the sea clutter has even managed to cover the banding gap of the ship target to some extent. White Gaussian noise can be added to make the simulated ISAR image, such as the one shown in Figure 51, further resemble the real ISAR image that is shown in Figure 1. Nevertheless, since the sea clutters are randomly generated, there is no guarantee that they will appear at every range bin with sufficient amplitude to cover the gap such as the one that is apparent in range bins 17 – 19 in Figure 50 and Figure 51.

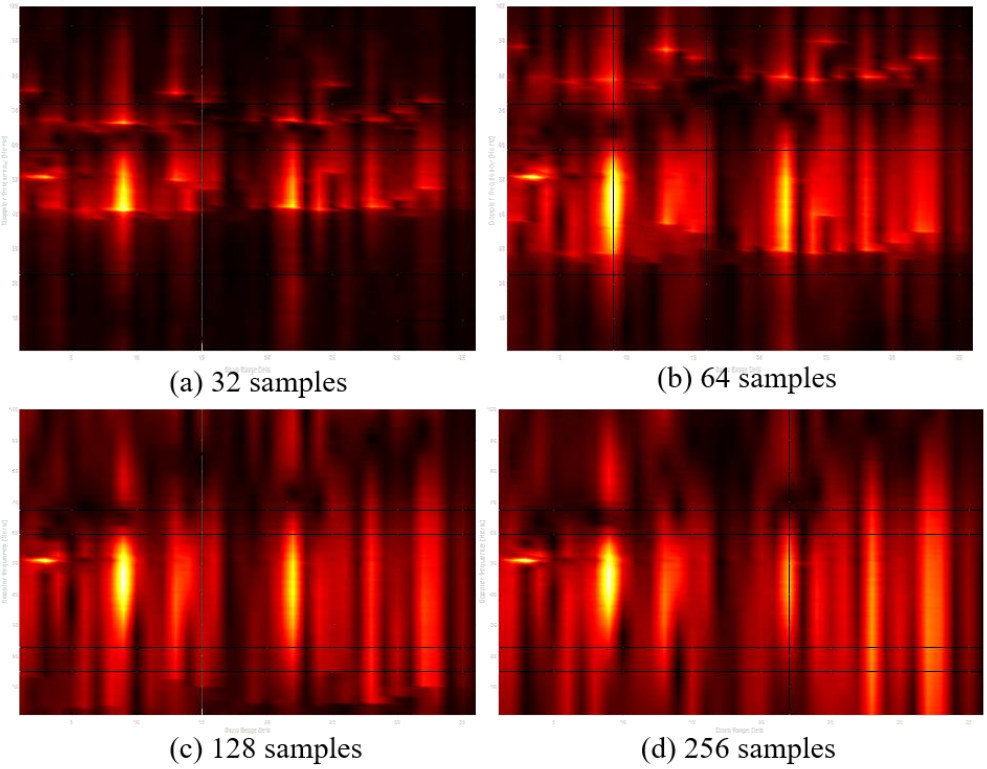


Figure 49. Range-Doppler Test Images for Sea Clutter Formed Using Different Numbers of Samples

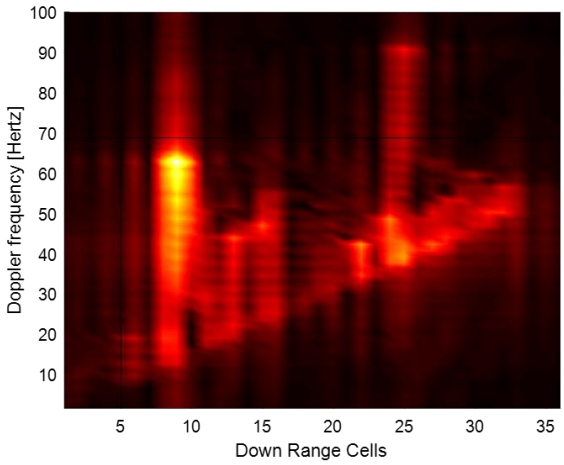


Figure 50. Test Image Showing the Use of Sea Clutters to Cover the Banding Gaps of the False Target

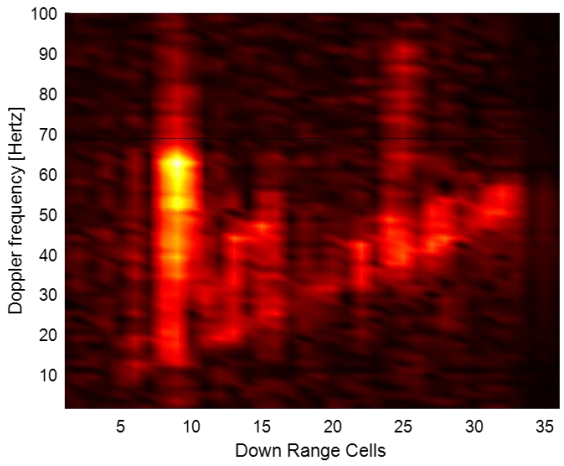


Figure 51. White Gaussian Noise Added to Figure 50

C. CHAPTER SUMMARY

The simulation of sea clutter for a high-resolution airborne radar was demonstrated in this chapter. Characteristics of the sea clutter range-Doppler profile were also discussed. The limitation of improving the fidelity of the DIS false target using sea clutter was also presented. Inevitably, the way ahead to improve the fidelity is to study the EM return of the ship target in a real-world environment. This recommendation is highlighted with other recommendations in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis has two objectives. The first objective was to design a phase converter and implement it in Verilog to support the ongoing development of the DIS. The second objective was to incorporate backscatter returns from the environment into the DIS false target image in order to improve its fidelity, thereby improving DIS effectiveness as an electronic protection solution for an ISAR.

In this thesis, a phase converter that utilized the CORDIC algorithm was designed using MATLAB and subsequently implemented in Verilog. The phase converter produced a five-bit phase result using nine-bit input I and Q data on each rising clock edge after an 18-clock pulse latency delay. Much time and effort were put into determining the number of iterations for the design so that the phase results would be accurate using a resolution of five bits. The phase converter also used a pipelined design that allowed the phase results to be produced on each clock cycle.

The creation of a sea clutter target profile for the DIS was also studied in this thesis. An existing sea clutter simulation model for the surface platform was adapted and modified to generate sea clutter for an airborne ISAR platform. The modified sea clutter model retained the ability in generating sea clutter at different sea states, waveform polarizations, and wind heading angle while using the normalized RCS model developed by NRL for a high grazing angle up to 60^{o} [17]. Using a random probabilistic model, we generated the sea clutter Doppler power spectrum to create the phase and gain coefficients for the sea clutter in the DIS. Since the sea clutter had a high Doppler resolution, the sea clutter maintained a convincing appearance as the ISAR increased its Doppler resolution.

In addition to the phase angle, the amplitude of the complex vector is another output from the phase converter, but it was not used in the DIS design. This piece of information is required to reconstruct the proper DRFM output power as shown in Figure 52. Since the amplitude is also scaled by a factor of 1.647, the CORDIC algorithm must be modified in order to have this scale factor taken into account.

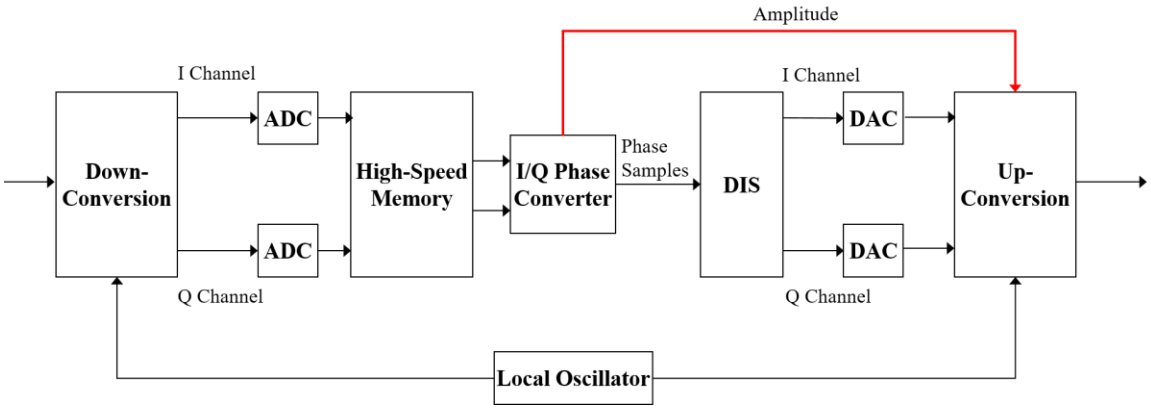


Figure 52. Use of the Amplitude Output of the Phase Converter to Reconstruct the ISAR Waveform

In this work, we evaluated the fidelity and image creation process of the DIS using a test pattern image. Further, insight into the bit-resolution influence on image quality was examined. Suggested future effort is to investigate a realistic image formation process.

Going forward, validation of the sea clutter model by collecting data in the Monterey Bay using a range-Doppler radar would aid in the creation of a realistic false target. The banding gaps described in this research were due to the finite resolution of the false target used to extract the imaging coefficients. A realistic false target can be generated by illuminating and collecting the ISAR waveform from a detailed target model using EM simulation software like CST Microwave Studio.

APPENDIX A. MATLAB CODES

A. CORDIC IMPLEMENTATION USING FLOATING POINT NUMBERS

```
clear all
clc
iter=8;
I =0;   % I signal
Q =0;   % Q signal

ideal_angle = atan2d(Q, I);
X = zeros(iter+1,1); % Xin
Y = zeros(iter+1,1); % Yin
Z = zeros(iter+1,1); % Zout;
d = zeros(iter+1,1); % direction vector
angleLUT = atand(2.^-(0:iter-1));

if (I>=0 && Q >=0) || (I>=0 && Q<0) %Quadrant 1 or 4
   X(1) = I;
   Y(1) = Q;
   Z(1) = 0;
elseif I<0 && Q>=0 % Quadrant 2
   X(1) = Q;
   Y(1) = -I;
   Z(1) = 90;
elseif I<0 && Q<0 % Quadrant 3
   X(1) = -Q;
   Y(1) = I;
   Z(1) = -90;
end

% rotate the vector for iter-number of times
for i = 1:iter
   if Y(i) < 0
      d(i) = 1;
   else
      d(i) = -1;
   end
   X(i+1) = X(i) - Y(i)*d(i)*2^-(i-1);
   Y(i+1) = Y(i) + X(i)*d(i)*2^-(i-1);
   if(X(i+1)==0 && Y(i+1)==0) %if the inputs are the origin points
     Z(i+1)=0;
   else
      Z(i+1) = Z(i) - d(i)*angleLUT(i);
```

```
    end
    fprintf('Iteration: %2d, Calculated angle after iteration: %7.3f, Rotated angle: %7.3f,
Error in degrees: %10g, Error in bits: %g\n',...
        [(i-1); Z(i);  - d(i)*angleLUT(i);(Z(i)-ideal_angle);log2(abs(Z(i)-ideal_angle))]);
end
fprintf('Iteration: %2d (Last), Rotated angle: %7.3f, Error in degrees: %10g, Error in bits:
%g\n',...
        [(i); Z(i+1);  (Z(i+1)-ideal_angle);log2(abs(Z(i+1)-ideal_angle))]);

for i = 1:iter
    fprintf('Iteration: %2d, Rotator magnitude: %g, Rotator scale factor: %g\n',...
    [i-1; sqrt(X(i)^2+Y(i)^2); sqrt(X(i)^2+Y(i)^2)/sqrt(I^2+Q^2)]);
end

    fprintf('Iteration: %2d (Last), Rotator magnitude: %g, Rotator scale factor: %g\n',...
    [i; sqrt(X(i+1)^2+Y(i+1)^2); sqrt(X(i+1)^2+Y(i+1)^2)/sqrt(I^2+Q^2)]);

figure (1)
cmap = colormap(lines(iter+3));
plot([0 I],[0 Q],'LineWidth', 2, 'Color',cmap(1,:))
legendInfo{1}=['Input Vector'];
hold on
plot([0 X(1)],[0 Y(1)],'LineWidth', 2, 'Color',cmap(2,:))
legendInfo{2}=['After initial rotation'];
for i=2:iter
    plot([0 X(i)],[0 Y(i)],'LineWidth', 2, 'Color',cmap(i+1,:));
    legendInfo{i+1}=['After rotation ' num2str(i-1)];
    hold on
end
plot([0 X(i+1)],[0 Y(i+1)],'LineWidth', 2, 'Color',cmap(i+2,:))
legendInfo{i+2}=['After rotation ' num2str(i), ' (Last)'];
yPos=0;
plot(get(gca,'xlim'), [yPos yPos], '-.k', 'LineWidth', 2);
legend(legendInfo)
hold off
grid on
xlabel('I')
ylabel('Q')
title('Vectoring Mode CORDIC Iterations')

figure(2)
plot(0:iter,Z(1:iter+1),'-dr','LineWidth', 2,'MarkerEdgeColor','k',...
    'MarkerFaceColor',[.49 1 .63],...
    'MarkerSize',10)
hold on
```

```
yPos_angle=ideal_angle;
plot(get(gca,'xlim'), [yPos_angle yPos_angle], '-.k', 'LineWidth', 2);
legend('Calculated angle', 'Actual angle')
hold off
xlabel('Iteration')
ylabel('Angle (degrees)')
title('Cumulative Angle Through Iterations')
```

B. CORDIC IMPLEMENTATION USING FIXED-POINT IMPLEMENTATION

```
clear all
clc

iter=18;      % # of iteration
inputbit = 9; % 1 sign bit 8 integer bit
%%%%%%%%%%%%%% I Q
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x = -230;
y = -1;
I = fi(x,1,inputbit,0);   % I is a 9-bit word with 1 sign bit and 8 integer bit
Q = fi(y,1,inputbit,0);   % Q is a 9-bit word with 1 sign bit and 8 integer bit
mag_ideal = abs(x+y*1i);
phase_ideal = atan2d(y, x);
%%%%%%%%%%%%%% X Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
frac_length = 18; % # fraction bit in X & Y
X = fi(zeros(iter+1,1),1,inputbit+2+frac_length,frac_length); % X array is a
9+frac_length+2(overflow) bit word with 1 sign bit, 10 integer bit and frac_length
fraction bit
Y = fi(zeros(iter+1,1),1,inputbit+2+frac_length,frac_length); % Y array is a
9+frac_length+2(overflow) bit word with 1 sign bit, 10 integer bit and frac_length
fraction bit

%%%%%%%%%%%%%% Z
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nbits = 5; % # bits in phase output
Z_fl = 17;   % # fraction bit in phase
phase_res = 360/2^nbits; %angle per bit
angleLUT = fi(atand(2.^-(0:iter-1))/phase_res,1,nbits+Z_fl,Z_fl);
Z = fi(zeros(iter+1,1),1,nbits+Z_fl+1,Z_fl); % Z array is a 5+1+Z_fl bit word with 1 sign
bit, 5 integer bit (1 integer is a guard bit for overflow) and Z_fl fraction bit
d = zeros(iter,1); % direction vector
%%%%%%%%%%%%%%%%%%%% Pre - rotation before
iteration%%%%%%%%%%%%%%%%%%
```

```
% if (I>0 && Q >0) || (I>0 && Q<0), no requirement to pre-rotate vector
% if I<0 && Q>0, rotate 90 degree clockwise
% if I<0 && Q<0 rotate 90 degree counterclockwise
if (I>=0 && Q >=0) || (I>=0 && Q<0)
   X(1) = I;
   Y(1) = Q;
   Z(1) = 0;
elseif I<0 && Q>=0
   X(1) = Q;
   Y(1) = -I;
   Z(1) = fi(90/phase_res,1,nbits+Z_fl,Z_fl);
elseif I<0 && Q<0
   X(1) = -Q;
   Y(1) = I;
   Z(1) = fi(-90/phase_res,1,nbits+Z_fl,Z_fl);
end

%%%%%%%%%%%%%%%%%%%% start of iteration
%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:iter
   if Y(i) < 0
      d(i) = 1;
      X(i+1) = X(i) - bitsra(Y(i), i-1);
      Y(i+1) = Y(i) + bitsra(X(i), i-1);
      Z(i+1) = Z(i) - angleLUT(i);
   else
      d(i) = -1;
      X(i+1) = X(i) + bitsra(Y(i), i-1);
      Y(i+1) = Y(i) - bitsra(X(i), i-1);
      if(X(i+1)==0 && Y(i+1)==0)
         Z(i+1)=0;
      else
         Z(i+1) = Z(i) + angleLUT(i);
      end
   end
end

%extract phase from the Z_array
a = bin(Z(iter+1)); %output in binary (integer + fraction bit)
b = a(length(a)-Z_fl+1); % 1st fraction bit (2^-1)
s = a(1);
c = a(2:length(a)-Z_fl); % phase output with only integer bit
if(bin2dec(b)==1)  % if b is '1' round up c by 1
   e = dec2bin(bin2dec(c)+ 1, length(c));
   if(bin2dec(e) == 32)
```

```
        e = dec2bin(0,length(c));
    end
else
    e = dec2bin(bin2dec(c), length(c));
end
f = bin2dec(e);

%extract magnitude from the X_array
mag_a = bin(X(iter+1)); %output in binary (integer + fraction bit)
mag_b = mag_a(length(mag_a)-frac_length+1); % 1st fraction bit (2^-1)
mag_c = mag_a(1:length(mag_a)-frac_length); % mag output with only integer bit
% if b is '1' round up c by 1
if(bin2dec(mag_b)==1)
    mag_e = dec2bin(bin2dec(mag_c)+ 1, length(mag_c));
else
    mag_e = dec2bin(bin2dec(mag_c), length(mag_c));
end
mag_f = bin2dec(mag_e);


fprintf('****Phase output*****\n');
fprintf('atand(y/x): %.12f\n', phase_ideal);
fprintf('Z output[before rounding]: %.12f (%s in binary) (%.12f degree)\n', (Z(iter+1)), a,
(Z(iter+1))*phase_res);
fprintf('phase error [before round]: %f\n', phase_ideal-((Z(iter+1))*phase_res));
fprintf('Z output [after rounding]: %d (%s in binary) (%.2f degree)\n', f, e, f*phase_res);
if(y<0 && f==0)
    fprintf('phase error [after rounding]: %f\n',phase_ideal-f*phase_res);
elseif(y<0)
    fprintf('phase error [after rounding]: %f\n',phase_ideal-(f*phase_res-360));
else
    fprintf('phase error [after rounding]: %f\n',phase_ideal-f*phase_res);
end
fprintf('****Magnitude output*****\n');
fprintf('amplitude: %.12f\n', mag_ideal);
fprintf('CORDIC magnitude output: %.12f (%s in binary)\n', X(iter+1), bin(X(iter+1)));
fprintf('ratio: %f\n', double(X(iter+1))/mag_ideal);
fprintf('CORDIC magnitude output after rounding: %d (%s in binary)\n', mag_f, mag_e);
fprintf('ratio (after rounding): %f\n',mag_f/mag_ideal);

figure (1)
cmap = colormap(lines(iter+3));
plot([0 I],[0 Q],'LineWidth', 2, 'Color',cmap(1,:))
grid on
legendInfo{1}=['Input Vector'];
```

```
hold on
plot([0 X(1)],[0 Y(1)],'LineWidth', 2, 'Color',cmap(2,:))
legendInfo{2}=['After initial rotation'];
for i=2:iter
    plot([0 X(i)],[0 Y(i)],'LineWidth', 2, 'Color',cmap(i+1,:));
    legendInfo{i+1}=['After rotation ' num2str(i-1)];
    hold on
end
plot([0 X(i+1)],[0 Y(i+1)],'LineWidth', 2, 'Color',cmap(i+2,:))
legendInfo{i+2}=['After rotation ' num2str(i), ' (Last)'];
yPos=0;
plot(get(gca,'xlim'), [yPos yPos], '-.k', 'LineWidth', 2);
legend(legendInfo)
hold off
xlabel('I')
ylabel('Q')
title('Vectoring Mode CORDIC Iterations')

figure(2)
if(x<0 && y<0)
    plot(0:iter,Z(1:iter+1)*phase_res,'-dr','LineWidth', 2,'MarkerEdgeColor','k',...
     'MarkerFaceColor',[.49 1 .63],...
     'MarkerSize',10)
else
    plot(0:iter,Z(1:iter+1)*phase_res,'-dr','LineWidth', 2,'MarkerEdgeColor','k',...
     'MarkerFaceColor',[.49 1 .63],...
     'MarkerSize',10)
end
hold on
yPos_angle=phase_ideal;
plot(get(gca,'xlim'), [yPos_angle yPos_angle], '--.k', 'LineWidth', 2);
legend('Calculated angle', 'Actual angle')
%yPos_angle_1=phase_ideal-360;
%plot(get(gca,'xlim'), [yPos_angle_1 yPos_angle_1], '-.b', 'LineWidth', 2);
legend('Calculated angle', 'Actual angle')%, 'Actual angle-360 degree')
hold off
xlabel('Iteration')
ylabel('Angle (degrees)')
title('Cumulative Angle Through Iterations')

A =[X, Y, Z, Z*phase_res,  abs(Z*phase_res-phase_ideal)]
B = [-d.*angleLUT',(-d.*angleLUT')*phase_res]
```

C. FINDING THE MAXIMUM PHASE ERROR

```
%this matlab code is used to find the maximum phase error and amplitude
%error for a range of iteration, xin and yin value

clear all
clc

format long
phasebit = 5; % # of bits in phase output
unit = 360/2^phasebit; %angle per bit

%%%%%%%%%%%%%% array for storing result
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cordic_phase_error = zeros(5,10); % store phase output of CORDIC
cordic_mag_error = zeros(2,10);   % store mag output of CORDIC
max_phase_error_location = zeros(1,3); % store the I Q for maximum errors


a = 1;
for iter = 8:8 % number of iteration required for CORDIC
   max_phase_error = 0;
   min_mag_scalefactor = 0;
   for i= -255:255     % range of I value 1-256 for 8 integer bits
     for q=-255:255   % range of Q value 1-256 for 8 integer bits
         ideal_angle=atan2d(q, i);   % ideal phase value
         J(i+256,q+256) = ideal_angle;
         mag_ideal=(sqrt(i^2 + q^2));  % ideal magnitude value
         [mag_cordic, phase_cordic] =cordic(i,q,iter); % call CORDIC function
         if(q<0 && phase_cordic==0)
           phase_error = ideal_angle - (phase_cordic*unit);
         elseif(q<0)
           phase_error = ideal_angle - (phase_cordic*unit-360);
         else
           phase_error = ideal_angle - (phase_cordic*unit);
         end
         K(i+256,q+256) = phase_cordic*unit;
         E(i+256,q+256) =  phase_error;

         mag_error =  mag_cordic/mag_ideal;

         % track the maximum error
         if (abs(phase_error)-max_phase_error> 0)
            max_phase_error = abs(phase_error);
            max_phase_error_location = [iter, i, q];
         end
```

```
            % track the scale factor
            if (abs(mag_error)-min_mag_scalefactor < 0)
               min_mag_scalefactor = abs(mag_error);
               max_mag_error_location = [iter, i, q];
            end
         end
      end
      cordic_phase_error(1:5,a) =
[iter,max_phase_error,log2(max_phase_error),max_phase_error_location(2),max_phase_
error_location(3)];
      cordic_mag_error(1,a) = iter;
      cordic_mag_error(2,a) = mag_error;
      cordic_mag_error(3, a) =max_phase_error_location(2);
      cordic_mag_error(4, a) =max_phase_error_location(3);
      a=a+1
end

% figure(2)
% CordicS = surf(-255:255, -255:255, K)
% set(CordicS,'LineStyle','none')
% xlabel('I')
% ylabel('Q')
% title('Arctangent(Q/I)- 8-Iteration CORDIC using Floating Point Precision
Calculation')

figure(3)
plot(reshape(J,[],1), reshape(abs(E), [],1), '.r', 'Linewidth', 0.1,'MarkerSize',0.1)
hold on
yPos=max(reshape(abs(E), [],1));
plot(get(gca,'xlim'), [yPos yPos], '-.k', 'LineWidth', 2);
xlabel('angle (degree)')
ylabel('error (degree)')
title('Angle Error of the CORDIC Algorithm')
legend('Error', 'Max Error')
ylim([0, yPos+0.3]);

function [mag, z] = cordic(x, y, iter)
   %%%%%%%%%%%%%%%%%%% # of bits for Input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   inputbit = 9; % word length, 1 sign bit the rest are integer bit

   %%%%%%%%%%%%%%%%%%%%%%%% phase output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   phasebit = 5; % # of bits in phase output
   Z_fl = 17; % # of fraction bit in phase
```

```
    frac_length = 18;
    unit = 360/2^phasebit; %angle per bit
    %%%%%%%%%%%%%%%%%%% input X Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    I = fi(x,1,inputbit,0);  % I signal
    Q = fi(y,1,inputbit,0);   % Q signal

    %%%%%%%%%%%% Define X Y Array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %fra_length = 8; %default 8
    X = fi(zeros(iter+1,1),1,inputbit+2+frac_length,frac_length); % X array is a
9+frac_length+1(overflow) bit word with 1 sign bit, 9 integer bit and frac_length fraction
bit
    Y = fi(zeros(iter+1,1),1,inputbit+2+frac_length,frac_length); % Y array is a
9+frac_length+1(overflow) bit word with 1 sign bit, 9 integer bit and frac_length fraction
bit

    %%%%%%%%%%%% Define Z angle Array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Z_fl = 1; %fraction length in phase output
    Z = fi(zeros(iter+1,1),1,phasebit+Z_fl+1,Z_fl); % Z array is a 5+Z_fl bit word with 1
sign bit, 4 integer bit and Z_fl fraction bit
    angleLUT = fi(atand(2.^-(0:20))/unit,1,phasebit+Z_fl,Z_fl); % create angle LUT
    %%%%%%%%%%%%%%%%%%%% Pre - rotation before
iteration%%%%%%%%%%%%%%%%%%%
    if (I>=0 && Q >=0) || (I>=0 && Q<0)
    X(1) = I;
    Y(1) = Q;
    Z(1)= 0;
    elseif I<0 && Q>=0
    X(1) = Q;
    Y(1) = -I;
    Z(1) = fi(90/unit,1,phasebit+Z_fl,Z_fl);
    elseif I<0 && Q<0
    X(1) = -Q;
    Y(1) = I;
    Z(1) = fi(-90/unit,1,phasebit+Z_fl,Z_fl);
    end

    %%%%%%%%%%%%%%%%%%%% start of iteration
%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:iter
        if Y(i) < 0
            X(i+1) = X(i) - bitsra(Y(i), i-1);
            Y(i+1) = Y(i) + bitsra(X(i), i-1);
```

```
            Z(i+1) = Z(i) - angleLUT(i);

        else
            X(i+1) = X(i) + bitsra(Y(i), i-1);
            Y(i+1) = Y(i) - bitsra(X(i), i-1);
            Z(i+1) = Z(i) + angleLUT(i);
            if(X(i+1)==0 && Y(i+1)==0)
                Z(i+1)=0;
            else
            Z(i+1) = Z(i) + angleLUT(i);
            end
        end
    end

    %extract phase from the Z_array

    overflow =0;
    b = bin(Z(iter+1)); %output in binary (integer + fraction bit)
    c = b(length(b)-Z_fl+1); % 1st fraction bit (2^-1)
    e = b(2:length(b)-Z_fl); % phase output with only integer bit
    % if c is '1' round up e by 1
    if(bin2dec(c)==1)
        f = dec2bin(bin2dec(e)+ 1, length(e));
        if(bin2dec(f) == 32)
            f = dec2bin(0,length(c));
        end
    else
        f = dec2bin(bin2dec(e), length(e));
    end
    z = bin2dec(f);
    %extract magnitude from the X_array
    mag_a = bin(X(iter+1)); %output in binary (integer + fraction bit)
    mag_b = mag_a(length(mag_a)-frac_length+1); % 1st fraction bit (2^-1)
    mag_c = mag_a(1:length(mag_a)-frac_length); % mag output with only integer bit
    % if b is '1' round up c by 1
    if(bin2dec(mag_b)==1)
        mag_e = dec2bin(bin2dec(mag_c)+ 1, length(mag_c));
    else
        mag_e = dec2bin(bin2dec(mag_c), length(mag_c));
    end
    mag = bin2dec(mag_e);
end
```

D. SEA CLUTTER SIMULATION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATLAB code for simulating sea clutter at different sea state,
% polarization and headwind direction
% Created by Owen Brooks and Modified by Dr. Sebastian Teich
% Adapted by Ang, Pak Siang to fit context of airborne operation
% Doppler resolution is determined by dp_pts
% Doppler and fluctuating power are saved as NuC1_rgdop and P_RC1_rgdop and
% exported to extract_para_shipv5.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%refresh
clear
close all
clc
set(0,'DefaultFigureWindowStyle','docked') %collects figures
%set(0,'DefaultFigureWindowStyle','normal') %collects figures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename=datestr(now,'yymmdd-HHMM-')

%% Radar Inputs & Design Parameters
c=3e8; %speed of light in m/2
dR=0.3; %m
dp_pts = 128; %number of pulse integration
prf = 200; %prf in DIS was 200
f_c = 10e9; %carrier frequency 10GHz
lambda = c/f_c; %wavelength
dF = 1/(dp_pts/prf); %doppler resolution = 1/T where T = Np/PRF Np = 128, PRF =200
dV = dF*2/lambda;
G_t=db2pow(32);     %assume 32dB
G_r=db2pow(32);     %assume 32dB
L=db2pow(3);        %total losses magnitude
P=500; % in Watts
B = 500e6; %bandwidth of chirp
fs_IF = 2*B;
ts=1/fs_IF; %sampling period
R_unamb = c*0.9995/2/prf; %unambiguous range = c(1-d)/(2xPRF)=150km
theta_el = 4.5 %+ 24.5; %elevation BW degrees
theta_az = 1.05; %azimuthal BW degrees

%Thermal Noise Level:
k=1.38e-23; %boltzman's (J/K)
```

```
T=290; %rx temp (K)
F=1; %noise factor, 1 is ideal
pn=k*T*B*F;

% Parameter inputs
r_e=4/3*6371e3;
hoe = 8000; % + 32000; % ht of aircraft in ft
HOE=hoe*0.3048;%m
range = 3000;% + 8000; %slant range m
f_D=2*000/lambda; %doppler shift due to aircraft's motion assuming 0. how about
100m/s?
graz = asin((HOE/range)+(HOE^2/(2*r_e*range))-(range/(2*r_e)));
graz_deg = rad2deg(graz);
a_c_length = range * deg2rad(theta_el)*csc(graz);
dR_h = dR*sec(graz);
mainbeam_rb = a_c_length/dR_h;
Rb = round(range/dR);
range_h = range*cos(graz);
Rb_min =Rb - floor((range_h - HOE/tan(graz+deg2rad(theta_el/2)))/dR_h);
Rb_max =round(mainbeam_rb)+Rb_min;
%rb_cmin =Rb - floor((range_h - HOE/tan(graz+deg2rad(theta_el/2)))/dR_h);
%rb_cmax =round(mainbeam_rb)+Rb_min;
%16 - for 32 range bin target
rb_cmin = floor(Rb - 256);
rb_cmax = round(Rb + 256);


SS=5; % seastate
Pol=2; % polarization V = 1 H = 2
ThWind=60; % headwind direction in angle

u=3.16*SS^0.8; %wind velocity m/s

if Pol==1
   Pol='V';
   K=0.18;
elseif Pol==2
   Pol='H';
   K=0.25;
end

%clutter inputs for each range bin occupied by false target:
Psi_c=zeros(1,rb_cmax - rb_cmin);
A_c=zeros(1,rb_cmax - rb_cmin);
sigma_0_c=zeros(1,rb_cmax - rb_cmin);
```

```
Psi_mainbeam=zeros(1,Rb_max - Rb_min);
A_mainbeam=zeros(1,Rb_max - Rb_min);
sigma_0_mainbeam=zeros(1,Rb_max - Rb_min);

%for the entire mainbeam
for i0=1:(Rb_max-Rb_min)

   %grazing angle, lower limit is zero
   Psi_mainbeam(i0)=asind(HOE/((i0+Rb_min)*dR)+HOE^2/(2*r_e*(i0+Rb_min)*dR)-
((i0+Rb_min)*dR)/(2*r_e)); %IEE
      if Psi_mainbeam(i0)<0
      Psi_mainbeam(i0)=0;
   end

   A_mainbeam(i0)=deg2rad(theta_az)*(dR*((i0+Rb_min)-
0.5))*dR*sec(deg2rad(Psi_mainbeam(i0))); %Nathanson

sigma_0_mainbeam(i0)=db2mag(NRL_SigmaSea(f_c/1e9,SS,Pol,Psi_mainbeam(i0),Th
Wind)); modelstring='NRL';
end

% for the 32 rangebins where the false target is expected to occupy
for i0=1:(rb_cmax-rb_cmin)

   %grazing angle, lower limit is zero
   Psi_c(i0)=asind(HOE/((i0+rb_cmin)*dR)+HOE^2/(2*r_e*(i0+rb_cmin)*dR)-
((i0+rb_cmin)*dR)/(2*r_e)); %IEE
      if Psi_c(i0)<0
      Psi_c(i0)=0;
   end

   A_c(i0)=deg2rad(theta_az)*(dR*((i0+rb_cmin)-0.5))*dR*sec(deg2rad(Psi_c(i0)));
%Nathanson

   % Sea Clutter RCS using NRL model
   sigma_0_c(i0)=db2mag(NRL_SigmaSea(f_c/1e9,SS,Pol,Psi_c(i0),ThWind));
modelstring='NRL';

end

%avg clutter power for entire mainbeam:
P_RC_0_mainbeam=zeros(1, Rb_max-Rb_min);
for i0=1:Rb_max-Rb_min %clutter in affected cells (within elevation beam)

   if SS==8; %Special case for NO clutter
```

```
        break
    end

    %empirical envelope expected from clutter within mainbeam

P_RC_0_mainbeam(i0)=P*G_t*G_r*lambda^2*sigma_0_mainbeam(i0)*A_mainbeam(i
0)/...
        ((4*pi)^3*(((i0+Rb_min)*dR)^4)*L); %friis
end

%avg clutter power for 32 range bins at the center where the target is placed:
P_RC_0=zeros(1, rb_cmax-rb_cmin);
for i0=1:rb_cmax-rb_cmin %clutter in affected cells (within elevation beam)

    if SS==8; %Special case for NO clutter
        break
    end

    %empirical envelope expected from RCS_0
    P_RC_0(i0)=P*G_t*G_r*lambda^2*sigma_0_c(i0)*A_c(i0)/...
        ((4*pi)^3*(((i0+rb_cmin)*dR)^4)*L); %friis
end


Nbar=0.01; %spike probability - varies w/ texture
step1=(5-2)/90; %ratio of spike power
step2=(40-2)/89;
rho(1:91)=fliplr(2:step1:5);
rho(92:181)=(2:step2:40);
%clutter doppler constants
Beta=2/lambda*(0.25+K*u)/11; Alpha=2/lambda*(0.25+K*u)-Beta; %10*Beta


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random clutter in main beam
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Texture_main=zeros(1,Rb_max-Rb_min);
spike_main=zeros(1,Rb_max-Rb_min);
n_spike_main=find(round((0.5/(1-Nbar))*rand(1,find(P_RC_0_mainbeam,1,'last'))));
spike_main(n_spike_main)=1;

mew_main=zeros(1,Rb_max-Rb_min);
sig1_main=abs(normrnd(0.2*u*cosd(ThWind)/lambda,20,1,Rb_max-Rb_min));
```

```
%fluctuations in main beam
for i0=1:Rb_max-Rb_min %vary the texture (slow moving) by range bin

Texture_main(i0)=KdistributionTexture(Psi_mainbeam(i0),A_mainbeam(i0),Pol,ThWind
,P_RC_0_mainbeam(i0));

mew_main(i0)=(Alpha+Beta*Texture_main(i0)/P_RC_0_mainbeam(i0))*cosd(ThWind)
+f_D;
end

fdl_main=find(Texture_main,1);
fdr_main=find(Texture_main,1,'last');
sig3_1_main=ceil(3*max(sig1_main')/dF);
maxsig1_main=2*ceil(3*max(sig1_main')/dF)+1;
NuC1_main=zeros(Rb_max-Rb_min,max(maxsig1_main));

for i0=fdl_main:fdr_main
      NuC1_main(i0,1:maxsig1_main)=[(round(mew_main(i0)/dF)-
sig3_1_main(1))*dF:dF:(round(mew_main(i0)/dF)+sig3_1_main(1))*dF];
end

P_RCi1_main=zeros(1,Rb_max-Rb_min);


for i0=fdl_main:fdr_main
   if spike_main(i0)==1

P_RCi1_main(i0)=exprnd((1+spike_main(i0)*rho(1+abs(wrapTo180(ThWind))))*Textur
e_main(i0))*Nbar+...%n=1
               exprnd(Texture_main(i0))*(1-Nbar);%n=0
   else
      P_RCi1_main(i0)=exprnd(Texture_main(i0))*(1-Nbar);%n=0
   end
end

P_RC1_main=zeros(Rb_max-Rb_min,max(maxsig1_main));

for i0=fdl_main:fdr_main
    for id=1:maxsig1_main(1)
      P_RC1_main(i0,id)=P_RCi1_main(i0)*dF/sig1_main(i0)/sqrt(2*pi)*exp(-
0.5*((NuC1_main(i0,id)-mew_main(i0))/sig1_main(i0))^2);
    end
end
```

```
% random value around normal distribution
P_RC1_main=normrnd(P_RC1_main,P_RC1_main/20);

figure(1001);
plot(((1:Rb_max-Rb_min)+Rb_min)*dR,pow2db(P_RCi1_main),'b+');hold on;
plot(((1:Rb_max-
Rb_min)+Rb_min)*dR,pow2db(P_RC_0_mainbeam),'Linewidth',3,'Color','r');
title(['Return power from the clutter within radar mainbeam for SS = ',num2str(SS),',
wind angle = ',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing angle =',
num2str(graz_deg), ' deg']);
legend('KA Distributed','Mean Power')
xlabel('Range (m)')
ylabel('Power (dBW)')

figure(1002);
subplot(211)
plot(((1:Rb_max-Rb_min)+Rb_min)*dR,pow2db(P_RCi1_main),'b+');hold on;
%plot(((1:Rb_max-Rb_min)+Rb_min)*dR,pow2db(P_RC1_main(:,1)'),'b+');hold on;
plot(((1:Rb_max-
Rb_min)+Rb_min)*dR,pow2db(P_RC_0_mainbeam),'Linewidth',3,'Color','r');
xlabel('Range (m)')
ylabel('Power (dBW)')
title(['clutter profile for SS = ',num2str(SS),', wind angle = ',num2str(ThWind),'deg,
',modelstring,' model, ', 'Grazing angle =', num2str(graz_deg), ' deg']);
hold off

subplot(212)
plot(((1:Rb_max-Rb_min)+Rb_min)*dR,NuC1_main(:,1),'b+'),hold on
plot(((1:Rb_max-Rb_min)+Rb_min)*dR,mew_main(1,:),'Linewidth',3,'Color','r')
grid on, xlabel('Range [m]'),ylabel('Doppler [Hz]')
title(['clutter & target(s) doppler for SS = ',num2str(SS),', wind angle =
',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing angle =', num2str(graz_deg), '
deg']);
hold off

RgDop_clutter_mainbeam = figure(1003); RgDop_clutter_mainbeam.Name='Range
Dopper - main beam'; Clutter.NumberTitle='off';%name figure in the window
figure(1003);
surf(NuC1_main, 1:Rb_max-Rb_min, (P_RC1_main), 'edgecolor', 'none');
xlabel('Doppler (Hz)'),ylabel('Range Bin'),zlabel('Power (W)')
title(['Range Doppler Distribution of Sea Clutter within the main beam for SS =
',num2str(SS),', wind angle = ',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing
angle =', num2str(graz_deg), ' deg']);
colorbar
%colormap(flipud(colormap))
```

```
figure(1004);
mesh(NuC1_main, 1:Rb_max-Rb_min, P_RC1_main, gradient(NuC1_main));
xlabel('Doppler (Hz)'),ylabel('Range Bin'),zlabel('Power (W)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random clutter in rangebin occupied by target:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Texture=zeros(1,rb_cmax-rb_cmin);
spike=zeros(1,rb_cmax-rb_cmin);
n_spike=find(round((0.5/(1-Nbar))*rand(1,find(P_RC_0,1,'last'))));
spike(n_spike)=1;

mew=zeros(1,rb_cmax-rb_cmin);
sig1=abs(normrnd(0.2*u*cosd(ThWind)/lambda,20,1,rb_cmax-rb_cmin));

%fluctuations in rangebin occupied by target
for i0=1:rb_cmax-rb_cmin %vary the texture (slow moving) by range bin
   Texture(i0)=KdistributionTexture(Psi_c(i0),A_c(i0),Pol,ThWind,P_RC_0(i0));
   mew(i0)=(Alpha+Beta*Texture(i0)/P_RC_0(i0))*cosd(ThWind)+f_D;
end

fdl=find(Texture,1);
fdr=find(Texture,1,'last');
sig3_1=ceil(3*max(sig1')/dF);
maxsig1=2*ceil(3*max(sig1')/dF)+1;
NuC1=zeros(rb_cmax-rb_cmin,max(maxsig1));

for i0=fdl:fdr
      NuC1(i0,1:maxsig1)=[(round(mew(i0)/dF)-
sig3_1(1))*dF:dF:(round(mew(i0)/dF)+sig3_1(1))*dF];
end

P_RCi1=zeros(1,rb_cmax-rb_cmin);


for i0=fdl:fdr
   if spike(i0)==1

P_RCi1(i0)=exprnd((1+spike(i0)*rho(1+abs(wrapTo180(ThWind))))*Texture(i0))*Nbar
+...%n=1
               exprnd(Texture(i0))*(1-Nbar);%n=0
   else
```

```
      P_RCi1(i0)=exprnd(Texture(i0))*(1-Nbar);%n=0
   end
end

P_RC1=zeros(rb_cmax-rb_cmin,max(maxsig1));

for i0=fdl:fdr
    for id=1:maxsig1(1)
      P_RC1(i0,id)=P_RCi1(i0)*dF/sig1(i0)/sqrt(2*pi)*exp(-0.5*((NuC1(i0,id)-
mew(i0))/sig1(i0))^2);
    end
end

% random value around normal distribution
P_RC1=normrnd(P_RC1,P_RC1/20);

%find the 512 center components of the doppler spectrum
% freq=zeros(1,512);
% for i0=1:(rb_cmax-rb_cmin)
%     [row, col] = find(abs(NuC1(i0,:)- mew(i0))< dF/2); %locate the indice where the
mean frequency is at
%     freq(i0) = (col);
%     NuC1_rgdop_512(i0, :) = NuC1(i0, freq(i0)-(512/2)+1:freq(i0)+512/2);
%     P_RC1_rgdop_512(i0,:) = P_RC1(i0, freq(i0)-(512/2)+1:freq(i0)+512/2);
% end
%find the 64 center components of the doppler spectrum
freq=zeros(1,64);
for i0=1:(rb_cmax-rb_cmin)
   [row, col] = find(abs(NuC1(i0,:)- mew(i0))< dF/2); %locate the indice where the mean
frequency is at
   freq(i0) = (col);
   NuC1_rgdop_64(i0, :) = NuC1(i0, freq(i0)-(64/2)+1:freq(i0)+64/2);
   P_RC1_rgdop_64(i0,:) = P_RC1(i0, freq(i0)-(64/2)+1:freq(i0)+64/2);
end
%find the 128 center components of the doppler spectrum
freq=zeros(1,128);
for i0=1:(rb_cmax-rb_cmin)
   [row, col] = find(abs(NuC1(i0,:)- mew(i0))< dF/2); %locate the indice where the mean
frequency is at
   freq(i0) = (col);
   NuC1_rgdop_128(i0, :) = NuC1(i0, freq(i0)-(128/2)+1:freq(i0)+128/2);
   P_RC1_rgdop_128(i0,:) = P_RC1(i0, freq(i0)-(128/2)+1:freq(i0)+128/2);
end
%find the 256 center components of the doppler spectrum
freq=zeros(1,256);
```

```
for i0=1:(rb_cmax-rb_cmin)
   [row, col] = find(abs(NuC1(i0,:)- mew(i0))< dF/2); %locate the indice where the mean
frequency is at
   freq(i0) = (col);
   NuC1_rgdop_256(i0, :) = NuC1(i0, freq(i0)-(256/2)+1:freq(i0)+256/2);
   P_RC1_rgdop_256(i0,:) = P_RC1(i0, freq(i0)-(256/2)+1:freq(i0)+256/2);
end


%plot the clutter, noise power samples at the rangebin where the target
%reside
Clutter=figure(4);Clutter.Name='Clutter Samples at the Range Bins where the Target
Reside';Clutter.NumberTitle='off';%name figure in the window
set(gcf, 'PaperPosition', [0 0 12.3333 8.5833])
subplot(211)
plot(((1:rb_cmax-rb_cmin)+rb_cmin)*dR,pow2db(P_RC_0),'Linewidth',3,'Color','r');
hold on
plot(((1:rb_cmax-rb_cmin)+rb_cmin)*dR,pow2db(P_RCi1),'b+'), hold on
plot(((1:rb_cmax-rb_cmin)+rb_cmin)*dR,pow2db(pn)*ones(1,rb_cmax-rb_cmin),'k-
.','Linewidth',3);
grid on, xlabel('Range [m]'),ylabel('Power [dBW]')
%axis([0,100*dR,min(min(pow2db(P_RC))),max(max(pow2db(P_RC)))])
title(['clutter & noise profile for SS = ',num2str(SS),', wind angle =
',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing angle =', num2str(graz_deg), '
deg']);
hold off

%plot 2D clutter sample spectrum at the range bin where the target reside
subplot(212)
plot(((1:rb_cmax-rb_cmin)+rb_cmin)*dR,NuC1(:,:),'b+'),hold on
plot(((1:rb_cmax-rb_cmin)+rb_cmin)*dR,mew(1,:),'Linewidth',3,'Color','r')
grid on, xlabel('Range [m]'),ylabel('Doppler [Hz]')
title(['clutter & target(s) doppler for SS = ',num2str(SS),', wind angle =
',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing angle =', num2str(graz_deg), '
deg']);
print(Clutter,['ClutterScatter, SS=',num2str(SS),', Pol=',Pol,', Th=',num2str(ThWind),',
Mod=',modelstring],'-dpng','-r300');
hold off

RgDop_clutter = figure(1005); RgDop_clutter.Name='Range Doppler 32 Rangebin';
Clutter.NumberTitle='off';%name figure in the window
surf(NuC1, 1:rb_cmax-rb_cmin, P_RC1, 'edgecolor', 'none');
xlabel('Doppler (Hz)'),ylabel('Range Bin'),zlabel('Power (W)')
```

```
title(['Range Doppler Distribution of Sea Clutter at Target for SS = ',num2str(SS),', wind
angle = ',num2str(ThWind),'deg, ',modelstring,' model, ', 'Grazing angle =',
num2str(graz_deg), ' deg']);
colorbar
%colormap(flipud(colormap))
```

E. NORMALIZED RCS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate Graph for Normalised RCS/RCS/Received Power vs Range/SS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%refresh
clear
close all
clc
set(0,'DefaultFigureWindowStyle','docked') %collects figures
%set(0,'DefaultFigureWindowStyle','normal') %collects figures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename=datestr(now,'yymmdd-HHMM-')

%% Radar Inputs & Design Parameters
c=3e8; %speed of light in m/2
dR=0.3; %m
dp_pts = 128; %number of doppler
prf = 200; %prf in DIS was 200
f_c = 10e9; %carrier frequency 10GHz
lambda = c/f_c; %wavelength
dF = 1/(dp_pts/prf); %doppler resolution = 1/T where T = Np/PRF Np = 128, PRF =200
dV = dF*2/lambda;
G_t=db2pow(32);      %assume 32dB
G_r=db2pow(32);      %assume 32dB
L=db2pow(3);        %total losses magnitude
P=500; % in Watts
B = 500e6; %bandwidth of chirp
fs_IF = 2*B;
ts=1/fs_IF; %sampling period
R_unamb = c*0.9995/2/prf; %unambiguous range = c(1-d)/(2xPRF)=150km
theta_el = 4.5 +24.5  ; %elevation BW degrees
theta_az = 1.05; %azimuthal BW degrees

%Thermal Noise Level:
```

```
k=1.38e-23; %boltzman's (J/K)
T=290; %rx temp (K)
F=1; %noise factor, 1 is ideal
pn=k*T*B*F;

% Parameter inputs
r_e=4/3*6371e3;
hoe = 5000 + 32000; % ht of aircraft in ft
HOE=hoe*0.3048;%m
range = 5000 + 12000; %slant range m
f_D=2*000/lambda; %doppler shift due to aircraft's motion assuming 0. how about
100m/s?
graz = asin((HOE/range)+(HOE^2/(2*r_e*range))-(range/(2*r_e)));
graz_deg = rad2deg(graz);
a_c_length = range * deg2rad(theta_el)*csc(graz);
dR_h = dR*sec(graz);
mainbeam_rb = a_c_length/dR_h;
Rb = round(range/dR);
range_h = range*cos(graz);
Rb_min =Rb - floor((range_h - HOE/tan(graz+deg2rad(theta_el/2)))/dR_h);
Rb_max =round(mainbeam_rb)+Rb_min;
sigma_0_mainbeam=zeros(6,Rb_max - Rb_min);
P_RC_0_mainbeam=zeros(6, Rb_max-Rb_min);
for SS=2:7; %sea state
   Pol=2; % polarization V = 1 H = 2
   ThWind=30; % headwind direction in angle
   u=3.16*SS^0.8; %wind velocity m/s

   if Pol==1
      Pol='V';
      K=0.18;
   elseif Pol==2
      Pol='H';
      K=0.25;
   end

      %clutter inputs for each range bin occupied by false target:
         Psi_mainbeam=zeros(1,Rb_max - Rb_min);
         A_mainbeam=zeros(1,Rb_max - Rb_min);

      % for the entire mainbeam
      for i0=1:(Rb_max-Rb_min)

         %grazing angle, lower limit is zero
```

```
Psi_mainbeam(i0)=asind(HOE/((i0+Rb_min)*dR)+HOE^2/(2*r_e*(i0+Rb_min)*dR)-
((i0+Rb_min)*dR)/(2*r_e)); %IEE
            if Psi_mainbeam(i0)<0
            Psi_mainbeam(i0)=0;
            end

        A_mainbeam(i0)=deg2rad(theta_az)*(dR*((i0+Rb_min)-
0.5))*dR*sec(deg2rad(Psi_mainbeam(i0))); %Nathanson
        main(i0)= deg2rad(theta_az)*(dR*((i0+Rb_min)-0.5));
        sigma_0_mainbeam(SS-
1,i0)=db2mag(NRL_SigmaSea(f_c/1e9,SS,Pol,Psi_mainbeam(i0),ThWind));
modelstring='NRL';
    end

    % avg clutter power for entire mainbeam:
    for i0=1:Rb_max-Rb_min %clutter in affected cells (within elevation beam)

        if SS==8; %Special case for NO clutter
            break
        end
        %empirical envelope expected from clutter within mainbeam
        P_RC_0_mainbeam(SS-1,i0)=P*G_t*G_r*lambda^2*sigma_0_mainbeam(SS-
1,i0)*A_mainbeam(i0)/...
            ((4*pi)^3*(((i0+Rb_min)*dR)^4)*L); %friis
    end

end

figure(1)
r = Rb_min*dR:dR:(Rb_max-1)*dR;
plot(r, pow2db(P_RC_0_mainbeam(1,:)))
ylabel('Received Power (dBW)')
xlabel('Range (m)')
grid on

figure(2)
plot(r, Psi_mainbeam(1,:))
ylabel('Grazing Angle(Degree)')
xlabel('Range (m)')
title(['Grazing Angle (Degree) vs Range']);
grid on

figure(3)
%plot(r, pow2db(A_mainbeam(1,:).*sigma_0_mainbeam(1,:)))
```

```
hold on
plot(r, pow2db(sigma_0_mainbeam(1,:)))
%plot(r, pow2db(dR*sec(deg2rad(Psi_mainbeam(1, :)))))
yyaxis right
ylabel('RCS(dbsm)')
%ylabel('Ac(dbsm)')
yyaxis left
ylabel ('Normalised RCS dbSM')
legend('RCS', 'normalized RCS')
hold off

xlabel('Range (m)')
title(['Area of Illuminated Patch vs Range']);
grid on
hold off

figure(4)
plot(r, pow2db(sigma_0_mainbeam(1,:)))
ylabel('Normalized RCS (dBsm)')
xlabel('Range (m)')
title(['Normalised RCS vs Range']);
grid on

figure(5)
plot(r, pow2db(A_mainbeam(1,:).*sigma_0_mainbeam(1,:)))
ylabel('RCS (dbsm)')
xlabel('Range (m)')
title(['RCS vs Range']);
grid on

figure(6)
for i0=1:6
   plot(r, pow2db(sigma_0_mainbeam(i0,:)))
   hold on
end
ylabel('Normalized RCS (dbsm)')
xlabel('Range (m)')
grid on
legend('SS=2','SS=3','SS=4','SS=5','SS=6','SS=7');

figure(7)
for i0=1:6
   plot(r, pow2db(P_RC_0_mainbeam(i0,:)))
   hold on
end
```

```
ylabel('Received Power (dBW)')
xlabel('Range (m)')
grid on
legend('SS=2','SS=3','SS=4','SS=5','SS=6','SS=7');
```

APPENDIX B. VERILOG CODES

A. CORDIC PROCESSOR

```
`timescale 1 ns/100 ps

//*********************************************************************************
//    Verilog code for CORDIC in vectoring mode.
//    Input a complex vector and output is amplitude and phase
//    Output phase is quantized to 5 bits. 0 deg = 5'd0, 11.25 deg = 5'd1
//    Size of Xout, Yout is 2 bit larger due to a system gain of 1.647 and overflow.
//
//     Modified by Ang, Pak Siang Jun 9 2017 based on previous version from by Kirk Weedman, KD7IRS, kirk@hdlexpress.com
//*********************************************************************************

module CORDIC (clock, Xin, Yin, Iout, Zout);

  parameter XY_SZ = 9;    // width of input and output data, 1 sign bit, rest are integer bit
  parameter phasebit = 5; // width of phase out data, 1 sign bit, rest are integer bit
  parameter iter = 18;    // number of iteration in CORDIC
  parameter mag_fb = 18;  // # of fraction bit introduced for X & Y array
  parameter phase_fb = 17; // # of fraction bit introduced for phase array

  input                  clock;
  input  signed  [XY_SZ-1:0]    Xin;    // # of bit for Xin is XY_SZ
  input  signed  [XY_SZ-1:0]    Yin;    // # of bit for Yin is XY_SZ
  output         [XY_SZ:0]      Iout;    // # of bit for Iout is XY_SZ+1 which cater for overflow
  output       [phasebit-1:0] Zout;    // # of bit for Zout is phasebit

  //-------------------------------------------------------------------------
  //                       arctan table
  //-------------------------------------------------------------------------

  // Note: The atan_table was chosen to be 12 bits wide giving resolution up to atan(2^-11)
  wire signed [phasebit+phase_fb-1:0] atan_table [0:18];
  // upper 1 bit = sign bit
  // 12'b010000000000 = 90 degrees
  // 12'b110000000000 = -90 degrees
  // upper 2 bits = 2'b00 which represents 0 - PI/2 range
```

```
  // upper 2 bits = 2'b01 which represents PI/2 to PI range
  // upper 2 bits = 2'b10 which represents PI to 3*PI/2 range (i.e. -PI/2 to -PI)
  // upper 2 bits = 2'b11 which represents 3*PI/2 to 2*PI range (i.e. 0 to -PI/2)
  // The upper 2 bits therefore tell us which quadrant we are in.
  assign atan_table[00] = 22'b0010000000000000000000; // 45.000 degrees -> atan(2^0)
  assign atan_table[01] = 22'b0001001011100100000001; // 26.565 degrees -> atan(2^-1)
  assign atan_table[02] = 22'b0000100111111011001110; // 14.036 degrees -> atan(2^-2)
  assign atan_table[03] = 22'b0000010100010001000100; // 7.12506 degrees -> atan(2^-
3)
  assign atan_table[04] = 22'b0000001010001011000011; // 3.576334 degrees ->
atan(2^-4)
  assign atan_table[05] = 22'b0000001010001011110110; // 1.789911 degrees ->
atan(2^-5)
  assign atan_table[06] = 22'b0000000101000101111110; // atan(2^-6)
  assign atan_table[07] = 22'b0000000001010001011111; // atan(2^-7)
  assign atan_table[08] = 22'b0000000000101000110000; // atan(2^-8)
  assign atan_table[09] = 22'b0000000000010100011000; // atan(2^-9)
  assign atan_table[10] = 22'b0000000000001010001100; // atan(2^-10)
  assign atan_table[11] = 22'b0000000000000101000110; // atan(2^-11)
  assign atan_table[12] = 22'b0000000000000010100011; // atan(2^-12)
  assign atan_table[13] = 22'b0000000000000001010001; // atan(2^-13)
  assign atan_table[14] = 22'b0000000000000000101001; // atan(2^-14)
  assign atan_table[15] = 22'b0000000000000000010100; // atan(2^-15)
  assign atan_table[16] = 22'b0000000000000000001010; // atan(2^-16)
  assign atan_table[17] = 22'b0000000000000000000101; // atan(2^-17)
  assign atan_table[18] = 22'b0000000000000000000011; // atan(2^-18)


  //-----------------------------------------------------------------------------
  //                          registers
  //-----------------------------------------------------------------------------

  //stage outputs
  reg signed [XY_SZ+1+mag_fb:0] X [0:iter+1];
  reg signed [XY_SZ+1+mag_fb:0] Y [0:iter];
  reg signed    [phasebit+phase_fb:0] Z [0:iter+1]; //

  //-----------------------------------------------------------------------------
  //                           stage 0
  //-----------------------------------------------------------------------------
  wire               [1:0] quadrant;
  assign   quadrant [1]= Xin[XY_SZ-1]; // = 1 if 1 is negative
  assign   quadrant [0]= Yin[XY_SZ-1];

  always @(posedge clock)
```

```
begin // make sure the rotation angle is in the -pi/2 to pi/2 range.  If not then pre-rotate
  case (quadrant)
    2'b00,
    2'b01:   // no pre-rotation needed for these quadrants
    begin    //
      X[0] <= Xin <<< mag_fb;
      Y[0] <= Yin <<< mag_fb;
      Z[0] <= 0;  //no need for any correction since vector is in the correct quadrant
    end

    2'b10:      //y>0, d=1
    begin
      X[0] <= Yin <<< mag_fb;
      Y[0] <= -Xin <<< mag_fb;
      Z[0] <= 22'b01000000000000000000; // add pi/2 from angle for this quadrant
    end

    2'b11:
    begin
      X[0] <= -Yin <<< mag_fb;
      Y[0] <= Xin <<< mag_fb;
      Z[0] <= 22'b11000000000000000000; // subtract pi/2 to angle for this quadrant
    end

  endcase
end

//-------------------------------------------------------------------------------
//                        generate stages 1 to iter
//-------------------------------------------------------------------------------
genvar i;

generate
for (i=0; i < iter; i=i+1)
begin: XYZ
  wire                 Y_sign;
  wire signed  [XY_SZ+mag_fb:0] X_shr, Y_shr;

  assign X_shr = X[i] >>> i; // signed shift right
  assign Y_shr = Y[i] >>> i;

  //the sign of the current rotation angle
  assign Y_sign = Y[i][XY_SZ+mag_fb+1]; // Y_sign = 1 if Y[i] < 0

  always @(posedge clock)
```

```
  begin
    // add/subtract shifted data
    X[i+1] <= Y_sign ? X[i] - Y_shr         : X[i] + Y_shr;
    Y[i+1] <= Y_sign ? Y[i] + X_shr         : Y[i] - X_shr;
    Z[i+1] <= Y_sign ? Z[i] - atan_table[i] : Z[i] + atan_table[i];
       end
end
endgenerate

//-------------------------------------------------------------------------------
//   Rounding for Z array: if the 1st fraction bit is '1', round up Z[iter] by 1
//-------------------------------------------------------------------------------
wire     MS_FB;
assign   MS_FB = Z[iter][phase_fb-1]; // extract the most significant fraction bit

always @(Z[iter])
begin //
  case (MS_FB)
    1'b0 :
       begin
         Z[iter+1] <= Z[iter][phasebit+phase_fb-1:phase_fb];
       end
    1'b1:
    begin
      Z[iter+1] <= Z[iter][phasebit+phase_fb-1:phase_fb]+1;
    end
   endcase
end

//-------------------------------------------------------------------------------
//  Rounding for X array: if the 1st fraction bit is '1', round up X[iter] by 1
//-------------------------------------------------------------------------------
wire     MS_FBx;
assign   MS_FBx = X[iter][mag_fb-1]; // extract the most signicant fraction bit


always @(X[iter])
begin //
  case (MS_FBx)
    1'b0 :
       begin
         X[iter+1] <= X[iter] >>> mag_fb;
       end
    1'b1:
    begin
```

```
          X[iter+1] <= (X[iter] >>> mag_fb)+1;
        end
      endcase
  end

  //---------------------------------------------------------------------------
  //    Check for Special Case : Xin = 0 Yin = 0 which will make PhaseOutput = 0
  //---------------------------------------------------------------------------

  always @(X[iter])
  begin
    if(X[iter] == 29'b00000000000000000000000000000)
          Z[iter+1] <= 5'b00000;
  end

  //---------------------------------------------------------------------------
  //                            output
  //---------------------------------------------------------------------------
  assign Iout = X[iter+1];
  assign Zout = Z[iter+1];

endmodule
```

B. TESTBENCH FILE

```
`timescale 1 ns/100 ps

module cordic_test;

localparam  SZ = 9; // bits of accuracy

reg signed [SZ-1:0] Xin, Yin;
wire  [4:0] Zout;
wire signed [SZ:0] Iout;
reg        clk;
reg signed [9:0] countx = 0;
reg signed [9:0] county = 0;
//

reg signed [63:0] i;
reg      start;

initial
```

```
begin: run

   $write("Starting sim");

   Xin =  -'d45;    //14'b00000000000000;
   Yin = 'd23;     //14'b00000000000000;
   #20;
   Xin = -'d45;    //14'b00000000000000;
   Yin = -'d23;     //14'b00000000000000;
   #20
   Xin =  -'d230;    //14'b00000000000000;
   Yin =  'd1;     //14'b00000000000000;


   //#400;
  // Xin = -'d45;
  // Yin = 'd23;


/*
// test for quandrant 1
   for (countx = 0; countx < 256; countx = countx + 1)
       begin
          for (county = 0; county < 256; county = county + 1)
         begin
            #20;
            Xin = countx;
         Yin = county;
         end
       end
*/
/*
// test for quandrant 2
   for (countx = -255; countx < 1; countx = countx + 1)
       begin
          for (county = 0; county < 256; county = county + 1)
         begin
            #20;
            Xin = countx;
         Yin = county;
         end
       end
*/
/*
// test for quandrant 3
```

```
   for (countx = -255; countx < 1; countx = countx + 1)
       begin
          for (county = -255; county < 1; county = county + 1)
        begin
           #20;
           Xin = countx;
        Yin = county;
        end
       end
*/
/*
// test for quandrant 4
   for (countx = 0; countx < 256; countx = countx + 1)
       begin
          for (county = -255; county < 1; county = county + 1)
        begin
           #20;
           Xin = countx;
        Yin = county;
        end
       end
*/
/*
// test  0 values should return 270 degrees
       for (county = -255; county < 0; county = county + 1)
        begin
           #20;
           Xin = 0;
        Yin = county;
        end
#100;
//  should return 90 degrees
          for (county = 1; county < 256; county = county + 1)
        begin
           #20;
           Xin = 0;
        Yin = county;
        end

#100;
// test  0 values should return 180 degrees
          for (countx = -255; countx < 0; countx = countx + 1)
        begin
           #20;
           Xin = countx;
```

```
        Yin = 0;
        end
#100;
//  should return 00 degrees
          for (countx = 1; countx < 256; countx = countx + 1)
        begin
           #20;
           Xin = countx;
        Yin = 0;
        end
*/


  #400;
  $write("Simulation has finished");
  $stop;
end

initial //generate clock
begin
          clk = 1'b0;
          forever
            #10 clk = ~clk;
end


CORDIC sin_cos (clk, Xin, Yin, Iout, Zout);


endmodule
```

LIST OF REFERENCES

[1] P. E. Pace, D. J. Fouts, S. Ekestorm, and C. Karow, "Digital false-target image synthesizer for countering ISAR," *IEEE Proc. - Radar, Sonar Navig*., 2002, vol. 149, no. 5, pp. 248–257.

[2] *Electronic Warfare and Radar Systems Engineering Handbook*, NAWCWD TP 8347, 4th ed., Naval Air Warfare Center Weapons Division, Point Mugu, CA, 2013. [Online]. Available: http://www.navair.navy.mil/nawcwd/ewssa/downloads/NAWCWD%20TP%208347.pdf

[3] D. J. Fouts, P. E. Pace, C. Karow, and S. R. T. Ekestorm, "A single-chip false target radar image generator for countering wideband imaging radars," *IEEE J. Solid-State Circuits*, vol. 37, no. 6, pp. 751–759, 2002.

[4] F. A. Le Dantec, "Performance analysis of a digital image synthesizer as a counter-measure against inverse synthetic aperture radar," M.S. thesis, Dept. Electron. Comp. Eng., Naval Postgraduate School, Monterey, CA, 2002.

[5] D. J. Fouts, P. E. Pace, "False target radar image generator for countering wideband imaging radars," U.S. Patent 6 624 780, Sep. 23, 2003.

[6] P. E. Pace, "Signal synthesizer and method therefor," U.S. Patent 2004/0201518, Oct. 14, 2004.

[7] P. E. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, 2nd ed. Norwood, MA: Artech House, 2009, p. 239.

[8] M. S. Dwan, "Test and evaluation of digital image synthesis technology for countering anti-ship capable missile threats," M.S. thesis, Dept. Electron. Comp. Eng., Naval Postgraduate School, Monterey, CA, 2007.

[9] Y. Lin, "Generation of clutter within a structured target synthesizer," M.S. thesis, Dept. Inform. Science., Naval Postgraduate School, Monterey, CA, 2012.

[10] L. Hongya and J. Xin, "Methods to recognize false target generated by digital-image-synthesiser," *Int. Symp. Inform. Sci. Eng.* 2008, vol. 1, pp. 71–75.

[11] G. Stimson, *Introduction to Airborne Radar*, 2nd ed. Mendham, NJ: SciTech Pub., 1998.

[12] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," *Proc. 1998 ACMSIGDA Sixth Int. Symp. F. Program. Gate Arrays FPGA 98*, 1998, pp. 191–200.

[13] R. C. Altmeyer, "Design, implementation, and testing of a VLSI high complex signal," M.S. thesis, Dept. Electron. Comp. Eng., Naval Postgraduate School, Monterey, CA, 2002.

[14] MathWorks. (n.d.). Calculate fixed-point arctangent. [Online]. Available: https://www.mathworks.com/examples/matlab-fixed-point-designer/mw/fixedpoint_product-fixpt_atan2_demo-calculate-fixed-point-arctangent. Accessed Jul. 15, 2017.

[15] K. Weedman. (2013, Mar., 31). Verilog tutorials. [Online]. Available: http://www.hdlexpress.com/Verilog/VT.html

[16] O. E. Brooks, "Submarine polyphase LPI radar design for surface search and track," M.S. thesis, Dept. Electron. Comp. Eng., Naval Postgraduate School, Monterey, CA, 2015.

[17] V. Gregers-Hansen and R. Mital, "An improved empirical model for radar sea clutter reflectivity," Naval Research Laboratory, Washington, DC, NRL/MR/5310–12-9346, Apr. 2012.

[18] B. R. Mahafza, *Radar Systems Analysis and Design Using MATLAB*, 3rd ed. Boca Raton, FL; London: CRC/Taylor & Francis, 2013.

[19] S. Watts, K. D. Ward, and R. J. A. Tough, "The physics and modeling of discrete spikes in radar sea clutter," *IEEE Proc.*, 2005, pp. 72–77. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1435796

[20] K. D. Ward, S. Watts, and R. J. A. Tough, *Sea Clutter: Scattering, the K-Distribution and Radar Performance*. London: Institution of Engineering and Technology, 2006.

[21] K. D. Ward and R. J. A. Tough, "Radar detection performance in sea clutter with discrete spikes," *IEEE Proc.*, 2002, pp. 253–257. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1174692

[22] S. Kemkemian, L. Lupinski, V. Corretja, R. Cottron, and S. Watts, "Performance assessment of multi-channel radars using simulated sea clutter," *IEEE Proc.*, 2015, pp. 1015–1020. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7131143

[23] S. Watts, "A new method for the simulation of coherent sea clutter," *IEEE Proc.*, 2011, pp. 052–057. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6324707

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California